

RL-TR-97-146
Final Technical Report
October 1997



HIGH PERFORMANCE OPTIMIZATION AND ABSTRACTION OF LARGE SIMULATION MODELS

University of Arizona

Bernard P. Zeigler and Yoonkeon Moon

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

DTIC QUALITY INSPECTED 4

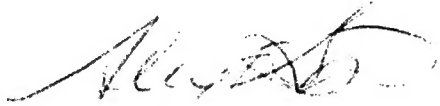
19980310 143

**Rome Laboratory
Air Force Materiel Command
Rome, New York**

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-146 has been reviewed and is approved for publication.

APPROVED:



ALEX F. SISTI
Project Engineer

FOR THE DIRECTOR:



JOSEPH CAMERA, Technical Director
Intelligence & Reconnaissance Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/IRAE, 32 Hangar Rd, Rome, NY 13441-4114. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE Oct 97	3. REPORT TYPE AND DATES COVERED Final Sep 95 - Dec 96		
4. TITLE AND SUBTITLE HIGH PERFORMANCE OPTIMIZATION AND ABSTRACTION OF LARGE SIMULATION MODELS		5. FUNDING NUMBERS C - F30602-95-C-0250 PE - 62702F PR - 4594 TA - 15 WU - N4		
6. AUTHOR(S) Bernard P. Zeigler and Yoonkeon Moon		8. PERFORMING ORGANIZATION REPORT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Arizona Dept of Electrical and Computer Engineering Tucson, AZ 85721		10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-97-146		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory/IRAE 32 Hangar Rd. Rome, NY 13441-4114		11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Alex F. Sisti, IRAE, 315-330-3983		
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) Modeling large scale systems with natural and artificial components requires storage of voluminous amounts of knowledge/information as well as computing speed for simulations to provide reliable answers in reasonable time. Computing technology is becoming powerful enough to support such high performance modeling and simulation. This report proposes a high performance simulation based optimization environment to support the design and modeling of large scale systems with high levels of resolution, and represents the results of contract F30602-95-C-0250, "Methodology for Simulation Model Abstraction."				
14. SUBJECT TERMS DEVs++, Optimization, Genetic Algorithms, Watershed Modeling, Mixed Resolution Modeling, Model Abstraction			15. NUMBER OF PAGES 130	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

This report was prepared under requirements for Contract F30602-95-C-0250 "Methodology for Simulation Model Abstraction". It reports results of research that was supported by this contract and that were based on earlier work funded under Rome Labs Contract F30602-95-C-0230 and the National Science Foundation.

TABLE OF CONTENTS

LIST OF FIGURES	5
LIST OF TABLES	8
ABSTRACT	9
1 INTRODUCTION	11
1.1 Problem Statement	11
1.2 The Layered High Performance Environment	14
1.3 Needs and Sources for High Performance	15
1.4 Report Organization	18
2 The DEVS Formalism: Basis for the Simulation Based Optimization Environment	19
2.1 Brief Review of the DEVS Formalism	19
2.2 Confluent Parallel DEVS	23
2.3 Implementation of Confluent DEVS-C++ Simulator	26
2.3.1 Specification and implementation of containers	28
2.3.2 DEVS C++ implementation over containers classes	30
3 The GA Searcher Layer	32
3.1 Genetic Algorithms	33
3.2 Distributed Asynchronous Genetic Algorithm	36
3.3 Experimental Results of the DAGA	43
3.3.1 Results for DeJong's Suite	45
3.3.2 Results for Rastrigin function	46
3.3.3 Results for Problem 5, 6 and 7	47
3.4 Application Example: Design of a Fuzzy Controller for the Inverted Pendulum	48
3.5 Implementation of GA C++	53
4 DEVS Modelling Example: Watershed	58
4.1 A Conceptual Hydrology Model for a Cell	59
4.2 DEVS representation of Infiltration	62
4.2.1 Green-Ampt Infiltration Model	63
4.2.2 DEVS Model for Infiltration	65

4.2.3	Implementing a DEVS Model using a Fuzzy System	71
4.2.4	Experimental Results	74
4.3	DEVS Representation of Runoff	78
4.4	Experimental Results	83
5	Optimization Example: Parameter Search for Watershed Models	89
5.1	Search for the parameters a and b	90
5.2	Search for the C range	92
5.3	Search for C surface roughness	97
5.4	Spatial Aggregation	102
5.4.1	Experimental Results	107
5.4.2	Multiresolution search based on parameter morphism	109
6	Conclusions	112
	Appendix A. Fuzzy Systems	116
	REFERENCES	119

LIST OF FIGURES

1.1	Layered representation of simulation based decision making	13
1.2	Search controlled high performance modelling and simulation environment.	15
2.1	Object-oriented implementation of DEVS on various platforms.	27
2.2	Implementation of DEVS using containers classes with C++	27
2.3	Five primitives of containers classes	28
2.4	Hierarchical construction of block models from atomic cell models	30
3.1	Construction of new population	34
3.2	Genetic operators: (a) crossover, and (b) mutation.	35
3.3	A typical procedure for executing a GA.	36
3.4	Asynchronous Genetic Algorithms	37
3.5	Operation flow of the AGA	38
3.6	Operation flow of the DAGA	39
3.7	Speedups of the AGA and DAGA on the CM-5.	40
3.8	GA controlled high performance simulation environment.	41
3.9	8 × 8 mesh graph with wraparound connections (the numbers inside circles show two solutions for the coloring problem).	45
3.10	The Inverted Pendulum	48
3.11	Initial conditions	49
3.12	Optimized membership functions and control rules.	50
3.13	(a) Pole angle, (b) Pole angular velocity, (c) Phase plane trajectories ending at (0,0), and (d) Input force.(Solid, dashed, and dotted curves correspond to initial conditions (10,20), (15,30), and (20,40), respectively).	51
3.14	(a) Pole length = 2.0 m, (b) Pole length = 0.5 m, (Solid, dashed, and dotted curves correspond to initial conditions (10,20), (15,30), and (20,40), respectively).	52
3.15	Implementation of GA C++.	54
3.16	GA C++ and Simulator classes.	55
3.17	Code example for GA C++ and Simulator.	56
4.1	Grid based representation of a watershed.	59
4.2	Conceptual hydrology model for a cell.	60
4.3	Connection of Cells	61

4.4	Green-Ampt infiltration model.	64
4.5	DEVS approximation model behavior (Solid and dashed curve represent the outputs of the DEVS and continuous model, respectively). 71	
4.6	The time to constant runoff $t_{to-const}$ for the silt soil (Dashed, solid and dotted curves are obtained for the fuzzy system, Newton-Raphson method and two-term Taylor series approximation, respectively). 75	
4.7	The membership functions of the fuzzy system (SM, ME, LA stand for small, medium and large, respectively).	76
4.8	State space partitioning for DEVS representation.	80
4.9	State space partitioning with granulized time axis.	83
4.10	Elevation map of target watershed.	84
4.11	Simulation results: hydrographs in the steady state.	84
4.12	The Number of events for DM(dt=0.00001), QM(D=0.1,d=0.01) and QM(D=50.0,d=0.01).	87
4.13	Accumulated total number of events in log scale (base 10).	88
5.1	Hydrographs of the optimized watershed model and KINEROS for the planes with the slopes of 0.01, 0.05 and 0.1.	93
5.2	Hydrographs of the optimized watershed model and KINEROS for a randomly generated rainfall event.	94
5.3	Hydrographs of the optimized watershed model and KINEROS for slope 0.01, 0.05 and 0.1.	95
5.4	Hydrographs of the optimized watershed model and KINEROS for randomly generated rainfall event.	96
5.5	A system that has some number of component models connected in cascade. (P_n is a parameter in component model n).	98
5.6	A system that has $n \times n$ blocks of models and each block has $m \times m$ models.	100
5.7	Parameter morphism.	103
5.8	Parameter morphism by GA optimization.	104
5.9	Spatial aggregation for one-dimensional flow.	105
5.10	Runoff of the base model and lumped models	109
5.11	Multiresolution search strategy.	110
6.1	Brown's pond elevation map.	113
6.2	Brown's pond runoff (m^3 /hour) after 2 simulated hours (1 hour after end of 1 hour long rainfall).	113
A.1	Fuzzy inference network and fuzzy subspaces	117

LIST OF TABLES

1.1	Iteration requirements for GA search	16
1.2	Orders of magnitude speedup and the corresponding amount of computation time that could be compressed into one minute	17
1.3	Sources of speedups in the high performance simulation based optimization environment	17
3.1	Execution times of Asynchronous Genetic Algorithm and Distributed Asynchronous Genetic Algorithm on the CM-5 (The unit is second).	40
3.2	Performance of the AGA and DAGA on DeJong's test suite (evals: average number of evaluations, std: standard deviation).	45
3.3	Performance of the AGA and DAGA on Problem 4 on the CM-5 with 400 nodes (solved: number of runs solved, avg: average best of 30 runs after 400,000 evaluations, avg. eval: average number of evaluations, total evals: total number of evaluations, total time: total execution time in seconds, time/eval: time taken per one evaluation in microseconds).	46
3.4	Performance of the AGA and DAGA on Problem 5 (solved: number of runs solved, avg. eval: average number of evaluations, std: standard deviation, problem size: problem size in bits).	47
3.5	Performance of the AGA and DAGA on Problem 6 (avg. eval: average number of evaluations, std: standard deviation).	47
3.6	Performance of the AGA and DAGA on Problem 7 (solved: number of solved, avg. eval: average number of evaluations, std: standard deviation, problem size: the size of mesh graph).	47
4.1	The execution time to optimize the fuzzy system on the CM-5 (measured for 1,000,000 evaluations).	77
4.2	Runoff of DEVS models in the Steady State. ($DM(dt)$ is the discrete time model, $QM(D, d)$ is the quantized DEVS model with quantum D and time granule, d , respectively).	85
4.3	Execution times of DEVS models.	86
5.1	Results of parameter search for a and b (optimization time is measured for 20,000 GA iterations on the CM-5 with 256 nodes).	91
5.2	Results of parameter search for C (although the best values are obtained within 1,200 GA iterations for all cases, the optimization time is measured for 20,000 iterations on the CM-5 with 256 nodes).	97

5.3	Execution times of watershed models on a Sparc-2 processor for a 2 hour rainfall event.	98
5.4	Results of global optimization (execution times are measured on the CM-5 with 256 nodes).	102
5.5	Results of global optimization (execution times are measured on a Sparc-1000).	102
5.6	Runoff error between base and lumped model for base model 1 (simulated for 3 hours with a 2 hour long 30 mm/hour rainfall event). 108	
5.7	Runoff error between base and lumped model for base model 2 (simulated for 3 hours with a 2 hour long 30 mm/hour rainfall event). 108	
5.8	Runoff error between base and lumped model for base model 3 (simulated for 3 hours with a 2 hour long 30 mm/hour rainfall event). 108	
5.9	Execution times of the base and lumped models on a Sparc-2 processor.109	

ABSTRACT

Modelling large scale systems with natural and artificial components requires storage of voluminous amounts of knowledge/information as well as computing speed for simulations to provide reliable answers in reasonable time. Computing technology is becoming powerful enough to support such high performance modelling and simulation. This report proposes a high performance simulation based optimization environment to support the design and modeling of large scale systems with high levels of resolution.

The proposed environment consists of three layers — modeling, simulation and searcher layer. The modeling layer employs the Discrete Event System Specification (DEVS) formalism and shows how it provides efficient and effective representation of both continuous and discrete processes in mixed artificial/natural systems necessary to fully exploit available computational resources. Focusing on the portability of DEVS across serial/parallel platforms, the simulation layer adopts object-oriented technology to achieve it. DEVS is implemented in terms of a collection of classes, called containers, using C++. The searcher layer employs Genetic Algorithms to provide generic, robust search capability. In this layer, a class of parallel Genetic Algorithms, called Distributed Asynchronous Genetic Algorithm (DAGA), is developed to provide the speed required for simulation based optimization of large scale systems.

This report presents an example of DEVS modeling for a watershed, which is one of the most complex ecosystems. The example shows a well-justified process of abstraction from traditional differential equation models to DEVS representation. An approach is proposed for valid aggregation of spatially distributed systems to reduce the simulation time of watershed models. DEVS representation and spatial aggregation assure relative validity and realism with feasible computational constraints. Throughout the report, several examples of GA optimization are presented to demonstrate the effectiveness of the proposed optimization environment in modeling large scale systems.

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Simulation-based design and testing before deployment has become the preferred way of fielding new systems in many areas. For example, simulation is to play a major role in the plans of the US Army in its restructuring for the information age[1]. The complexity of behavior that modern systems can exhibit demands computing power far exceeding that of current workstation technology. To address challenging computing problems using high-resolution, large scale representations of systems composed of natural and artificial elements, high performance simulation-based design environments are characterized by two levels of intensive knowledge/information processing. At the decision-making level, searches are conducted through vast problem spaces of alternative design configurations and associated model structures; at the execution level, simulations generate and evaluate complex candidate model behaviors, possibly interacting with human participants in real time.

This report proposes a high performance simulation based optimization environment to support modelling and simulation of large-scale system with natural and artificial components at high levels of resolution. The basic modelling formalism

employed is that of discrete events for representing both continuous and discrete processes. We will show that discrete event representations have significant performance and conceptual advantages over that of discrete time systems for continuous dynamic systems.

Figure 1.1 depicts simulation-based decision making, in terms of a layered system of functions. In this paradigm, decision makers, for example, forest managers, base their decisions on experiments with alternative strategies (e.g., for reducing the risk of wild fires) where the best strategies (according to some criteria) are put into practice. For a variety of reasons, experiments on models are preferred to those carried out in reality. For realistic models (e.g., of forest fire spread), such experiments can not be worked out analytically and require direct simulation. The design of our environment to support all these activities is based on the layered collection of services shown in Figure 1.1, where each layer uses the services of lower layers to implement its functionality. To provide generic robust search capability we employ Genetic Algorithms (GAs) as the searcher in the model space. The optimization layer employs the searcher to find good or even optimal system designs (models).

Experience with this environment has shown that only large numbers of interconnected processing nodes can provide 1) the memory to hold the enormous amounts of knowledge/information necessary to model complex systems, and 2) the simulation speed required to provide reliable answers in reasonable time. Currently one can marshal such large numbers of computing nodes dedicated to a single problem only

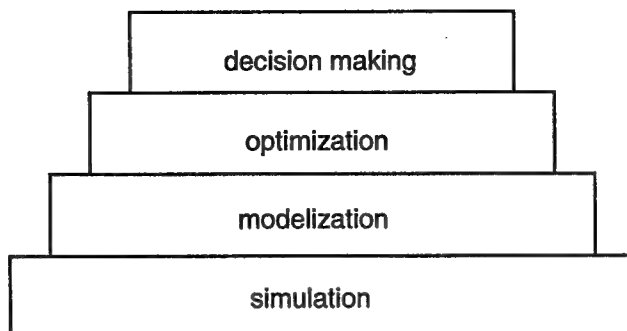


Figure 1.1: Layered representation of simulation based decision making

in scalable, high performance platforms such as the Connection Machine, CM-5 or the IBM SP2 which contain up to 1024 processors. However, we will show that at least a million fold increase in either speed or numbers of nodes is needed for such systems to support optimization of large scale models. Unfortunately, the cost of such platforms is beyond the means of most potential users and there are only a small number accessible in national high performance computing centers.

By contrast, the numbers and speeds of desktop computers (PCs and workstations) are escalating rapidly so that harnessing these resources might offer a solution. However the obstacles to networking large numbers of distributed computing resources are formidable. One survey result indicates that the largest network cluster contains 130 workstations connected with Parallel Virtual Machine (PVM) over Ethernet[2] , much less than the massively parallel computer platforms. One significant social barrier to dedicating large numbers of workstations to a single computation is distributed ownership which tends to discourage shared usage. Two technical barriers which we addressed in the design of our environment are heterogeneity and portability.

1.2 The Layered High Performance Environment

As illustrated in Figure 1.2, in the proposed environment the various processes are executed concurrently within a heterogeneous, distributed computing environment. Each GA agent has access to a simulator for executing its experiments. Although the simulator is shown as a single entity, it too could be distributed among the processors. Generally an experiment consists of several trials testing how well a particular intelligent control (supervisory or management) agent functions in a prescribed problem environment. This environment is represented as a simulation model which is controlled/observed by the agent through an appropriate experimental frame. The model in each simulator may actually be one several related models at several levels of abstraction ranging from low to high resolution. The GA may initially search through the coarser space spanned by the most abstract model before going on to higher resolution searches [3].

As an example, the family of model abstractions could be discrete event models of a watershed varying in resolution. The experimental frame may provide a storm track as input and it might observe the resulting flooding pattern. The effectiveness of a set of pre-flood stage sensors as an early warning system might then be reported to a GA agent and manipulated by the distributed GA to search for improving the locations for placement of the sensors.

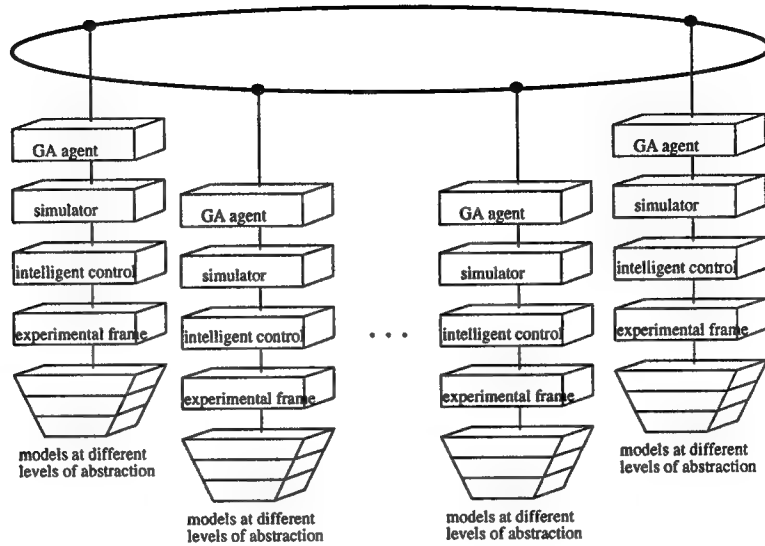


Figure 1.2: Search controlled high performance modelling and simulation environment.

1.3 Needs and Sources for High Performance

The demands of such an environment on any technology capable of supporting it are enormous. Realistic simulations of large models with decision-making components are time consuming. The GA searcher, although robust, is apt to require thousands of simulation evaluations to locate an optimal configuration.

For example we have demonstrated a successful application of our simulation-based optimization environment to fuzzy control system synthesis [4], to 1024-node parameter search problems in optical interconnection network design[5] and to watershed modelling [6]. The number of iterations required to identify the optimum in these cases is shown in Table 1.1.

Optimization Run	Iterations Needed
Fuzzy Control System Synthesis	100,000
Watershed Modelling	1,000,000
Optical Interconnection Network Design	10,000,000

Table 1.1: Iteration requirements for GA search

These optimizations were run on the CM-5, but for illustration purposes, suppose that they had been executed on a conventional workstation. Were each simulation to require 10 minutes, 1,000,000 GA iterations would require a good part of person's lifetime. High performance is clearly needed to speed up such computations. Given our finite lifetimes and immediate daily concerns, the time we are willing to wait for its answers is strictly bounded. One would be hard pressed, for example, to find an existing simulation run taking years of continuous computation. However, with sufficient increases in speed afforded by high performance, system design studies that are not feasible today could be undertaken. Table 1.2 translates the effect of orders of magnitude speed up on feasibility of such computation. For example, with a speedup of 1,000 times the just mentioned example of an optimization can be done in days instead of years. With another factor of 1,000 in speedup, optimizations that now take decades (and so are infeasible) would become commonplace. As a consequence, there could be a tremendous increase in the reliability, safety and effectiveness of tomorrow's complex systems such as flood, hurricane warning systems, forest fire fighting robotic systems, or space-based reconnaissance systems.

Speedup Order of Magnitude	Computation Time Reduced to 1 min
1	minute
100	hours
1,000	days
10,000	months
100,000	years
1,000,000	decades
10,000,000	centuries

Table 1.2: Orders of magnitude speedup and the corresponding amount of computation time that could be compressed into one minute

Speedups of the order of 1,000,000 are in fact attainable with the technology and methodology on the horizon. Table 1.3 shows where such performance improvements could come from in the simulation environment. We will provide evidence for a) up to 1,000 fold speedup gained by properly mapping continuous models into efficient DEVS approximations, b) up to 1,000 speedup with the application of parallel/distributed processing at the GA search level. The upper bound of this estimate is based on the best performance achievable on an N processor system, where N is currently around a thousand (e.g. 1024 in the CM-5). The number of processors in a single platform will increase another order of magnitude with the construction of the 9,000 processor system announced by Intel. By multiplying the two speedup factors, the speedup possible is of the order of 10^6 .

Speedup Order	Sources
100 – 1,000	DEVS Representation
100 – 1,000	Parallel/Distributed GA Search

Table 1.3: Sources of speedups in the high performance simulation based optimization environment

We will offer some evidence for the attainability of the individual speedups in Table 1.3 in the sequel.

1.4 Report Organization

First we review the DEVS formalism which is the basic mathematical language of the proposed environment and shows the implementation of the DEVS simulator in Chapter 2. Chapter 3 introduces a new parallel GA, called Distributed Asynchronous GA, employed in the searcher layer of the proposed environment. This chapter also includes some test results of the DAGA and shows an application example, designing a fuzzy controller. In Chapter 4 we discuss the use of DEVS in efficient continuous process representation taking watershed modeling as an example. In Chapter 5 we show an example of parameter search in the proposed environment for watershed models developed in Chapter 4. Finally Chapter 6 concludes the report.

CHAPTER 2

The DEVS Formalism: Basis for the Simulation Based Optimization Environment

To discuss the performance advantages of discrete event model formulations we will need to review the modelling formalism, called DEVS, underlying the current high performance simulation based optimization environment.

2.1 Brief Review of the DEVS Formalism

We now review the basic concepts of the DEVS formalism and its associated simulation methodology.

In the conceptual framework underlying the DEVS formalism[7], the modelling and simulation enterprise concerns four basic objects:

- the *real system*, in existence or proposed, which is regarded as fundamentally a source of data
- the *model*, which is a set of instructions for generating data comparable to that observable in the real system. The structure of the model is its set of instructions. The behavior of the model is the set of all possible data that can be generated by faithfully executing the model instructions.

- the *simulator* which exercises the model's instructions to actually generate its behavior.
- *experimental frames* capture how the modeller's objectives impact model construction, experimentation and validation. Experimental frames are formulated as model objects in the same manner as the models of primary interest. In this way, model/experimental frame pairs form coupled model objects which can be simulated to observe model behavior of interest.

The basic objects are related by two relations:

- the *modelling relation*, linking real system and model, defines how well the model represents the system or entity being modelled. In general terms a model can be considered valid if the data generated by the model agrees with the data produced by the real system in an experimental frame of interest.
- The *simulation relation*, linking model and simulator, represents how faithfully the simulator is able to carry out the instructions of the model.

The basic items of data produced by a system or model are *time segments*. These time segments are mappings from intervals defined over a specified time base to values in the ranges of one or more variables. The variables can either be observed or measured.

The structure of a model may be expressed in a mathematical language called a *formalism*. The discrete event formalism focuses on the changes of variable values

and generates time segments that are piecewise constant. Thus an event is a change in a variable value which occurs instantaneously. In essence the formalism defines how to generate new values for variables and the times the new values should take effect. An important aspect of the formalism is that the time intervals between event occurrences are variable in contrast to discrete time where the time step is a fixed number.

Independence from a fixed time step affords important advantages for modelling and simulation. Multiprocess models contain many processes operating on different time scales. Such models are difficult to describe when a common time granule must be chosen on which to represent them all. Moreover, simulation is inherently inefficient since the states of all processes must be updated in step with this smallest time increment – such rapid updating is wasteful when applied to the slower processes. In contrast, in a discrete event model every component has its own control over the time of its next internal event. Thus, components demand processing resources only to the extent dictated by their own intrinsic speeds or their responses to external events.

DEVS falls within the formalisms identified by Ho[8] for discrete event dynamical systems (DEDS). Work on a mathematical foundation of discrete event dynamic modeling and simulation began in the 70s[7, 9, 10] when DEVS was introduced as an abstract formalism for discrete event modeling. Because of its system theoretic basis, DEVS is a universal formalism for discrete event dynamical systems (DEDS)[11].

Indeed, DEVS is properly viewed as a short-hand to specify systems whose input, state and output trajectories are piecewise constant[11]. The step-like transitions in the trajectories are identified as discrete events.

Discrete event models provide a natural framework to include discrete formalisms for intelligent systems such as neural nets, fuzzy logic, qualitative reasoning, and expert systems. However, traditional differential equation models continue to be the basic paradigm for representing the physical environments in which intelligent agents operate. We have proposed that DEVS-based systems theory, incorporating discrete and continuous subformalisms, provides a sound, general framework within which to address modelling, simulation, design, and analysis issues for natural and artificial systems[12].

The universality claims of the DEVS just cited are addressed by characterizing the class of dynamical systems which can be represented by DEVS models. Praehofer and Zeigler[13] showed that any causal dynamical system which has piecewise constant input and output segments can be represented by DEVS. We call this class of systems *DEVS-Representable*[11]. In particular, Differential Equation Specified Systems (DESS) are often used to represent both the system under control and the controller, which, as a decision making component, has a natural DEVS representation.

DEVS supports construction of new models by interconnecting already existing models as components. Such interconnection, called coupling, is specified in a well defined manner embodied in the formalism of the *coupled model*[9]. Closure under coupling guarantees that coupling of class instances results in a system in the same class. The class of DEVS-representable dynamical systems is closed under coupling[14, 13]. Closure is an essential property since it justifies **hierarchical, modular construction** of both DEVS models and the (continuous or discrete) counterpart systems they represent.

2.2 Confluent Parallel DEVS

The DEVS formalism, as revised to enable full exploitation of parallel execution[15] is the basis for the DEVS-C++ high performance environment under construction.

A DEVS basic model is a structure:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta, \rangle$$

X : a set of input events.

S : a set of sequential states.

Y : a set of output events.

$\delta_{int} : S \rightarrow S$: internal transition function.

$\delta_{ext} : Q \times X^b \rightarrow S$: external transition function,

X^b is a set of bags over elements in X ,

$\delta_{con} : S \times X^b \rightarrow S$: confluent transition function.

$\lambda : S \rightarrow Y^b$: output function.

$ta : S \rightarrow R_{0+\rightarrow\infty}$: time advance function,

where $Q = \{(s, e) | s \in S, 0 < e < ta(s)\}$,

e is the elapsed time since last state transition.

DEVS models are constructed in a hierarchical fashion by interconnecting components (which are DEVS models). The specification of interconnection, or coupling, is provided in the form of a coupled model. The structure of such a *coupled model* is given by:

$$DN = \langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} \rangle$$

X : a set of input events.

Y : a set of output events.

D : an index set for the components of the coupled model.

For each i in D ,

M_i is a component DEVS model.

For each i in $D \cup \{self\}$, I_i is the set of influencees of i .

For each j in I_i ,

$Z_{i,j}$ is a function,

the i -to- j output translation mapping.

The structure is subject to the constraints that for each i in D ,

$M_i = \langle X_i, S_i, Y_i, \delta_{inti}, \delta_{exti}, \delta_{coni}, ta_i \rangle$ is a DEVS basic structure,

I_i is a subset of $D \cup \{self\}$, i is not in I_i ,

$Z_{self,j} : X_{self} \rightarrow X_j$,

$Z_{i,self} : Y_i \rightarrow Y_{self}$,

$Z_{i,j} : Y_i \rightarrow X_j$.

Here *self* refers to the coupled model itself and is a device for allowing specification of external input and external output couplings. More explicitly, I_{self} is the set of components that receive external input; also if *self* is in I_i , then component i 's output appears as external output of the coupled model.

The behavior of a coupled model is constructed from the behaviors of its components and the coupling specification. The *resultant* of a coupled model is the formal expression of such behavior. Closure of the formalism under coupling is demonstrated by constructing the resultant and showing it to be a well defined DEVS. As already stated, such closure ensures that hierarchical construction is well defined since a coupled model (as represented by its resultant) is a DEVS model that can be

coupled with other components in a larger model. Details of closure proof are given by Chow[15]

2.3 Implementation of Confluent DEVS-C++ Simulator

In designing a DEVS-based high performance simulation based optimization environment, our goal was portability of models across platforms at a high level of abstraction. A DEVS model should not have to be rewritten to run on serial, parallel or distributed environment. Ideally, this invariance should apply at the high level of abstraction (set-theory) in which DEVS is formulated. However, a computational equivalent of this level does yet not exist (although efforts are beginning in that direction). Falling short of this ideal, but still significant, is the ability to port DEVS models written in the same computer language across platforms. There are numerous advantages to such portability. To name several that are especially important in this context: 1) models developed on a serial workstation, with all its comfortable development support, can be easily ported after verification to a parallel system for high performance production runs, 2) in a parallel/distributed environment, the same form of model description can be used for the interaction of model components executing within the (serial) nodes as for the (parallel) interaction of components executing on different nodes (more of this later).

Object-oriented technology is the key to achieving DEVS portability objectives while retaining the flexibility to mitigate concomitant performance costs. Perhaps the

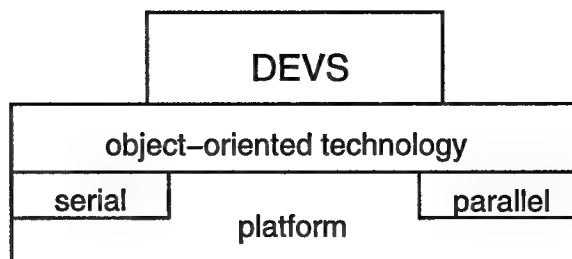


Figure 2.1: Object-oriented implementation of DEVS on various platforms.

most characteristic attribute of this technology is its ability to separate behavior from implementation, enabling distinct implementations of the same behavior to coexist [16]. As shown in Figure 2.1, DEVS is implemented in an object-oriented form which enables it to be executed on serial or parallel platforms. The DEVS formalism is expressed as objects and their interactions with the details of the implementation (serial or parallel) hidden within the objects. The user interacts with only those interfaces that manifest the DEVS constructs while being shielded from the ultimate execution environment.

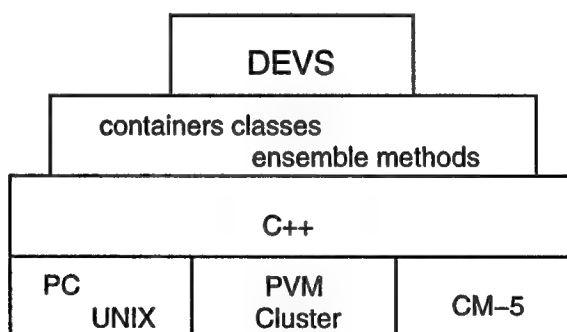


Figure 2.2: Implementation of DEVS using containers classes with C++

The approach is illustrated in greater detail in Figure 2.2. Due to its rapidly growing availability, C++ was employed as the target object-oriented language. As

shown, DEVS is implemented in terms of a collection of classes, called *containers*. In their usual serial guise, such classes provide well-known means for defining list data structures and their manipulation. However, a more abstract and useful characterization of their functionality is that containers provide services to group objects into collections and coordinate the activity within such groups.

2.3.1 Specification and implementation of containers

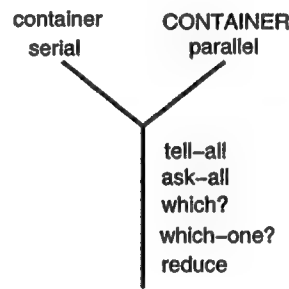


Figure 2.3: Five primitives of containers classes

This is illustrated in Figure 2.3 which enumerates five basic primitives for coordinating behavior of objects in a container[16]. In outline:

- *tell-all* sends the same command to each object in a container.
- *ask-all* sends the same query to each object and returns a container holding the responses (which are also objects).
- *which?* returns the subcontainer of all objects whose response to a boolean query is TRUE.

- *which-one?* returns one of the objects in the container whose response to a boolean query is TRUE.
- *reduce* aggregates the responses of the objects in a container to a single response (e.g., taking the sum).

While these so-called *ensemble methods* may seem more parallel than sequential in nature, they have abstract specifications that are independent of how one chooses to implement them. Thus, using the polymorphism properties of C++ we define two classes for each abstract container class; one (lower-case) implementing the ensemble methods in serial form, the other (upper-case), implementing them in parallel form (Figure 2.3). The serial implementations run on any architecture that has a C++ compiler. In particular, if the nodes of a parallel or distributed system run C++, then the serial containers will work on them. However, the implementation of parallel CONTAINERS involves physical (as opposed to virtual) message passing among objects residing on different nodes. Such message passing must be implemented within the communications primitives afforded by the parallel/distributed system in question. For example, massively parallel CM-5 implementation employs CMMD(CM-5 message passing library). Likewise, a network of workstations linked together under PVM [17] offers the communication primitives supplied by PVM.

In the next section, we discuss the serial implementation of the DEVS C++ simulator based on containers (refer to [18] for the parallel implementation of DEVS C++).

2.3.2 DEVS C++ implementation over containers classes

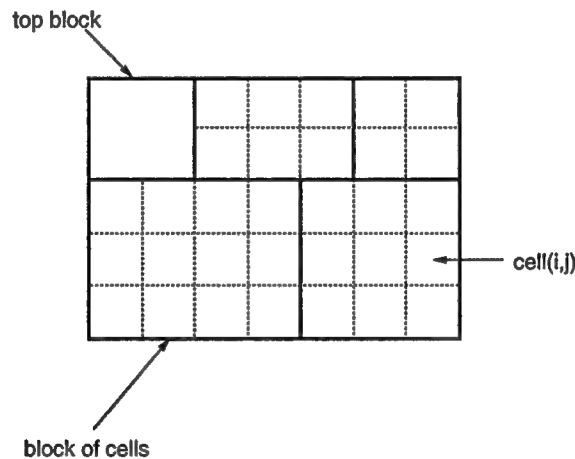


Figure 2.4: Hierarchical construction of block models from atomic cell models

To illustrate, consider a two-dimensional grid of cells as shown in Figure 2.4. The cells could be the atomic components in a landscape model of watershed in Chapter 4. They are grouped into blocks. The closure property of DEVS guarantees that each block can itself be regarded as a DEVS model which can now be considered as a component model. These components are then grouped together to form a new DEVS model which is equivalent in behavior to the original.

Blocks (or digraph models) are effectively containers whether they contain cells (or atomic models) or lower level blocks (or digraph models). In terms of ensemble methods, a cycle of DEVS simulation in block models can be outlined as follows:

1. Compute the global next event time, tN : use *reduce* to get the minimum of component times to next event,
2. Tell all components the global tN and if a component is imminent (tN equals to global tN), then generate output messages (using λ),
3. Sort and distribute (using coupling) output messages,
4. Tell all components: if a component is imminent (tN equals to global tN) or has incoming mail (external events) or both, then execute transition functions. If a component is both imminent and has incoming mails (inputs), it executes the confluent transition function (δ_{con}). If a component is imminent and has no incoming mail, it executes the internal transition function (δ_{int}). If a component is not imminent and has incoming mails, then it executes the external transition function (δ_{ext}).

CHAPTER 3

The GA Searcher Layer

A major advantage of high resolution models is that many of the parameter values needed to calibrate the model to its real system counterpart are obtained directly from available engineering or measurement-derived data. Still, a large simulation model typically has many more parameters that are unknown. These parameters need adjustment to tune the model to real world observed behavior or to optimize its performance to achieve a desired objective. Searching through such large parameter spaces for optimal, or even acceptable points is a daunting task, especially in multiple process models where each simulation run may require hours or days to complete. The more that automated optimizers can relieve human modelers of this search task, the faster will be the pace of advance in the modelling or design effort. Therefore, optimization-based control of simulation is a key feature of our high performance environment.

In this chapter we introduce a new GA, called Distributed Asynchronous Genetic Algorithm(DAGA), which is employed as a searcher in the high performance modelling and simulation environment.

3.1 Genetic Algorithms

GAs are a class of stochastic operators that successively transform an initial population of individuals until a convergence criterion is met [19, 20, 21, 22, 23, 24, 25]. Each individual represents a candidate system design and each design is evaluated using the underlying simulation layer to give some measure of its *fitness*. On each iteration, a new population is formed by selecting fitter individuals and transforming them in hopes of obtaining ones even fitter. Typically, *crossover* creates new children by combining parts from two parent individuals. *Mutation* creates new individuals by introducing small random changes [22]. After some number of generations the search converges and is successful if the best individual represents the global optimum. GAs often outperform classical optimization methods in search, optimization and learning [26, 23, 27, 22]. Interest has increased in their potential application to modeling, simulation and design of complex real world systems.

The first step in the implementation of GAs is to generate an initial population randomly in most cases. Each member of this population will be a binary string of length L which corresponds to the problem encoding. Each string is sometimes referred to as a *genotype* or a *chromosome*. On the execution of the GA, it starts with the *current population*. The initial population becomes the current population in the beginning or in the first generation (the term *generation* will be explained shortly). Selection is applied to the current population to create an intermediate population.

Then recombination and mutation are applied to the intermediate population to create the *next population*. The process of going from the current population to the next population is called one generation in the execution of a GA. Now the next population becomes the current population for the next generation and repeats the above process until a member of the current population represents a solution to the given problem. Figure 3.1 shows how one generation is constructed in the execution of a GA.

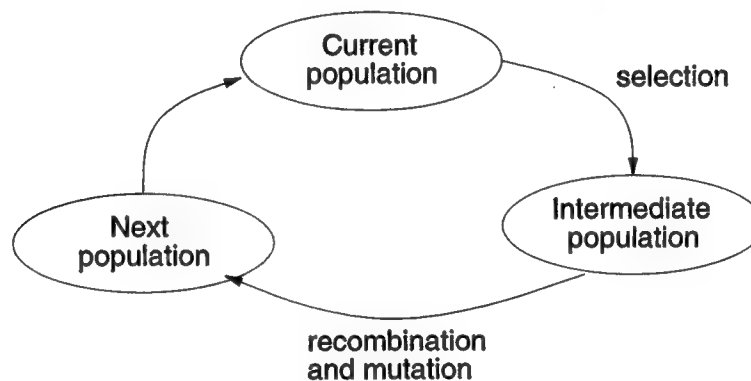


Figure 3.1: Construction of new population

The construction of the intermediate population from the current population is carried out as follows. In the first generation the initial population is considered as the current population. After calculating *fitness* of each member (string) in the current population, selection is performed. The fitness is a metric which measures, in most cases, how close the member is to the solution of the given problem in the search space. Thus, the design of a function to measure fitness is very problem dependent. After selection has been carried out, the construction of the intermediate population

is complete and recombination can occur. This can be viewed as creating the next population from the intermediate population. Recombination is done using a genetic operator called *crossover*. Crossover is applied to paired strings with a probability of p_c as follows. Pick a pair of strings in the intermediate population. With probability of p_c , recombine these strings to form two new strings, and then insert them in the next population. Figure 3.2.a shows how a crossover operator works on paired strings called parent strings to generate new strings called offsprings.

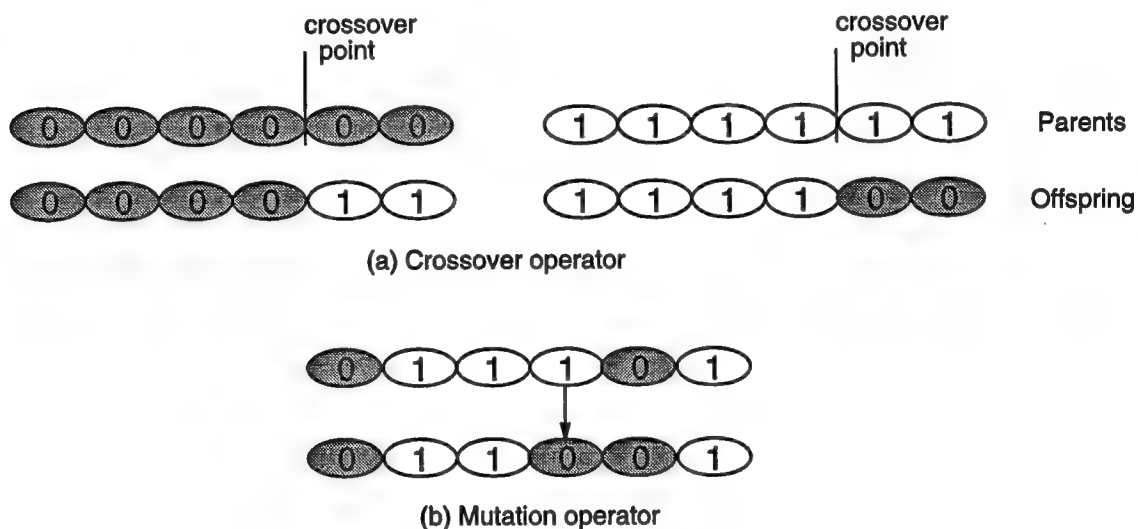


Figure 3.2: Genetic operators: (a) crossover, and (b) mutation.

After recombination, we can also apply another genetic operator called *mutation* as shown in Fig. 3.2.b as follows. For each bit in a bit string, mutate (or complement a bit in the string) with some probability p_m .

After the process of selection, recombination, and mutation is complete, the next population can be evaluated. The process of evaluation, selection, recombination,

and mutation forms one generation in the execution of a GA. Thus, GAs use the notion of survival of the fittest by passing good genes (potential solutions) to the next generation and combining different genes to explore new search points. In summary, Fig. 3.3 describes typical steps for executing GAs.

```
A genetic algorithm
{
  Initialize population;
  Current_population = Initial_population;
  while termination criterion not reached;
  {
    Evaluate Current_population;
    Select members in Current_population for Intermediate_population;
    Perform crossover and mutation for Next_population;
    Current_population = Next_population;
  }
}
```

Figure 3.3: A typical procedure for executing a GA.

3.2 Distributed Asynchronous Genetic Algorithm

Adapted to the high performance simulation based optimization environment, GAs intelligently generate trial model candidates for simulation-based evaluation. Although schemes exist for parallelizing GAs [20, 21, 28, 29, 30], we designed a new parallel GA, called *Distributed Asynchronous Genetic Algorithms (DAGAs)*, which is particularly suited to the demands of the high performance simulation environment. The following is an overview of the DAGA adapted to the optimization layer for the proposed environment.

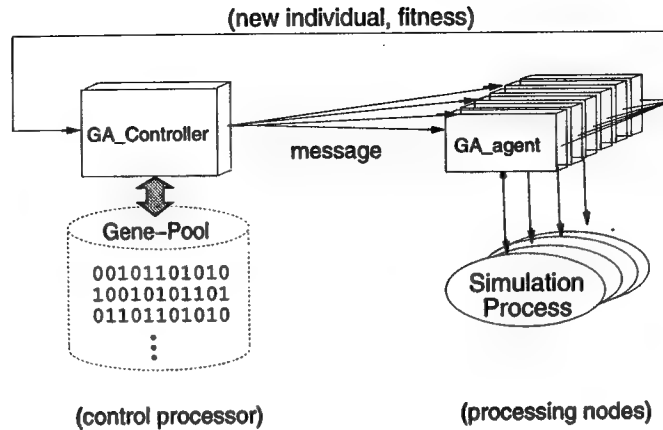


Figure 3.4: Asynchronous Genetic Algorithms

The DAGA is an extension of the Asynchronous Genetic Algorithm (AGA) [30]. The AGA maintains the genetic operations in one processing node (GA-controller) and distributes the evaluation (simulation) processes to many nodes (GA-agents) as shown in Figure 3.4. The operation of the AGA is shown in Figure 3.5.

The novelty of the AGA is that the GA-controller doesn't wait for a full generation to complete, which would severely reduce throughput when simulation times are widely dispersed [30]. However, experimental results (refer Section 3.3) show two drawbacks of the AGA as follows:

1. The AGA fails to solve some class of problems, that is, it is not robust.
2. Due to centralization of the GA-controller, the communication overhead and sequential genetic operations bottleneck processing as the number of processing nodes increases.

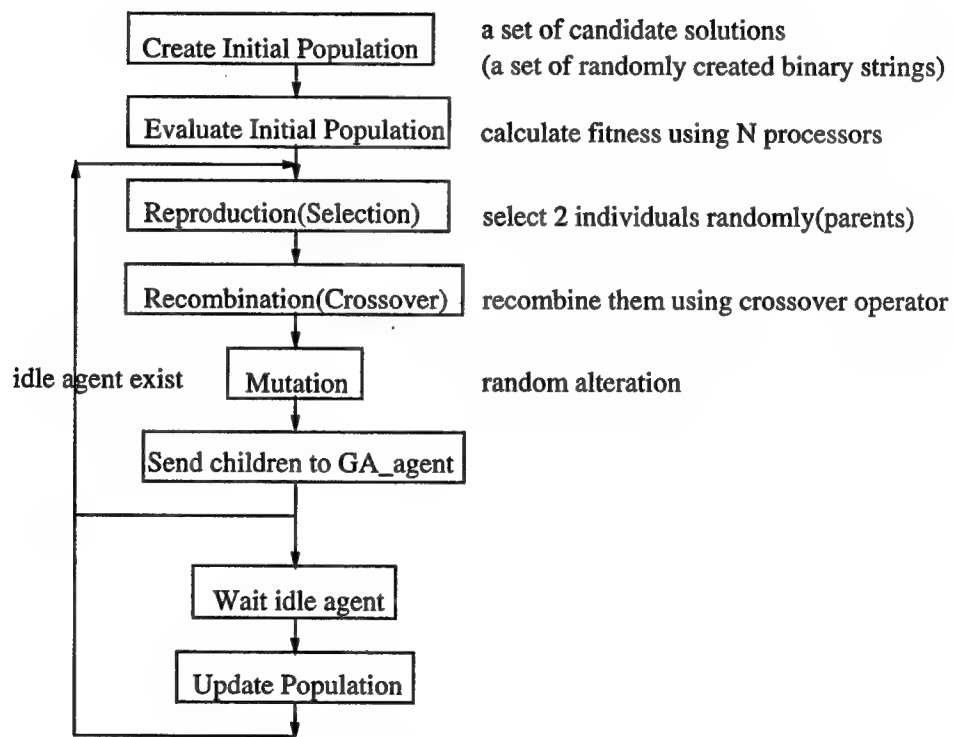


Figure 3.5: Operation flow of the AGA

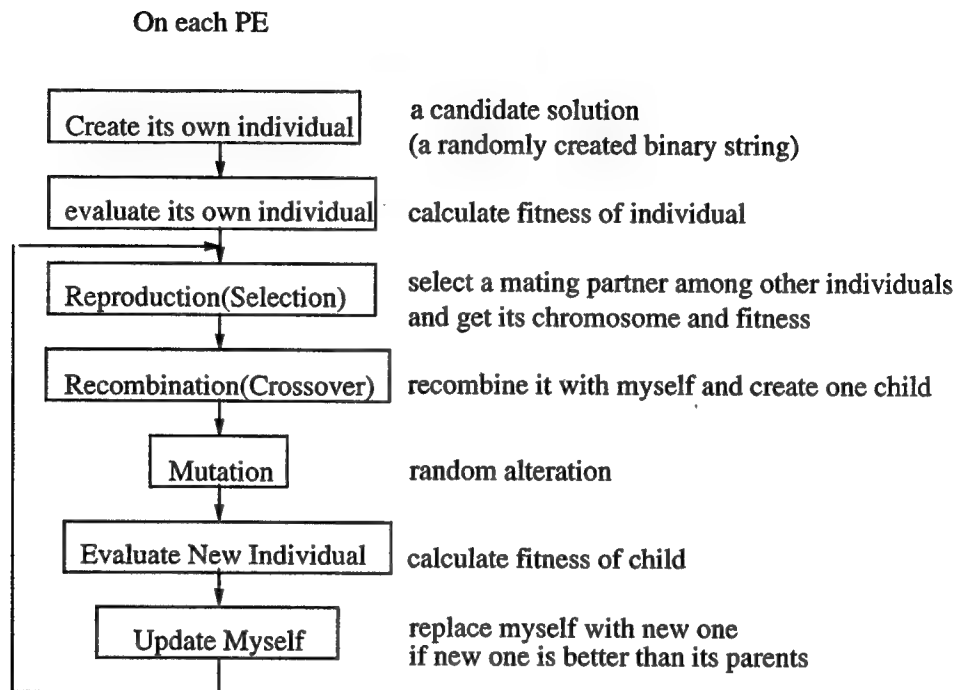


Figure 3.6: Operation flow of the DAGA

To solve the first problem, we employ the different selection and updating schemes in the DAGA. To achieve scalability (solve the second problem), the DAGA distributes both evaluation processes and genetic operations to processing nodes as shown in Figure 1.2. The operation of the DAGA is outlined as shown in Figure 3.6.

As with the original asynchronous scheme, GA agents do not wait for a full generation to complete, which would severely reduce throughput when simulation times are widely dispersed. Moreover, in this scheme there is no central processing to bottleneck performance since all genetic operations are carried out by processors autonomously with at most minimal exchange with a randomly chosen partner.

# of PEs	1	32	64	128	256	512
AGA	14,410	770	512	282	254	259
DAGA	12,940	567	298	142	71	36

Table 3.1: Execution times of Asynchronous Genetic Algorithm and Distributed Asynchronous Genetic Algorithm on the CM-5 (The unit is second).

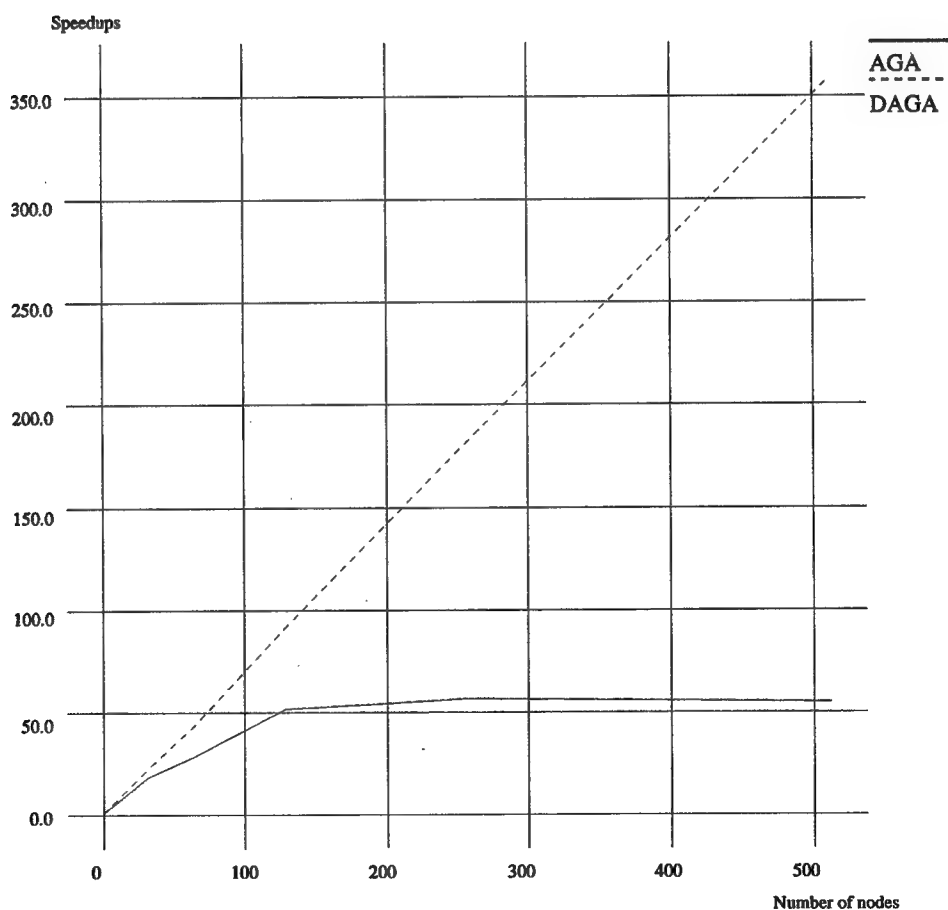


Figure 3.7: Speedups of the AGA and DAGA on the CM-5.

The beneficial effect is shown in Table 3.1 and Figure 3.7 which compare the DAGA with the earlier scheme (the AGA) applied to the same problem of optimizing a fuzzy controller design for an inverted pendulum (refer to Section 3.4) in terms of execution time. Note that while the original scheme's performance does not scale with increasing processors, the distributed version achieves quite close to a linear speedup depending on number of processors.

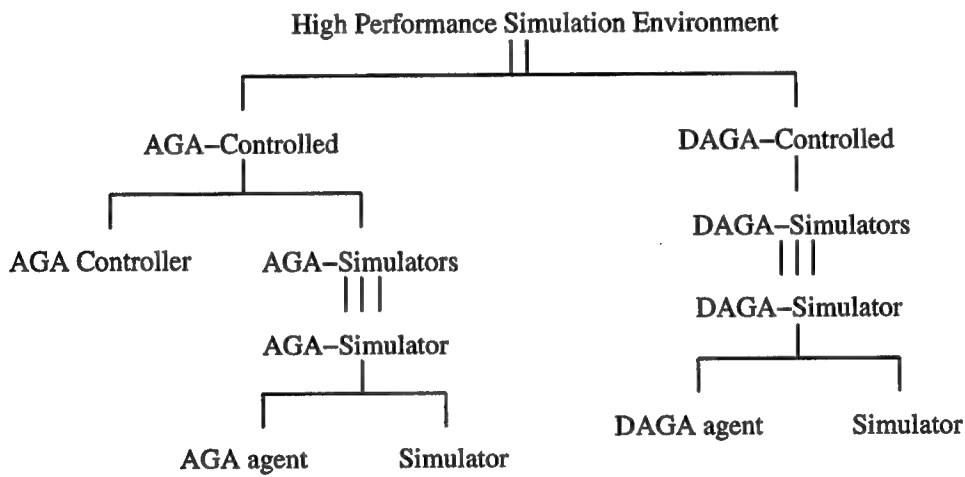


Figure 3.8: GA controlled high performance simulation environment.

The DAGA can also be implemented like the AGA as follows: The GA controller

1. creates its initial population randomly (population initialization step),
2. evaluates the generated individual's fitness using GA agents,
3. (as soon as this initial evaluation is completed) *randomly* selects two individuals (parents),
4. applies crossover and mutation operators,

5. randomly selects one child and evaluates the selected child using an idle GA agent,
6. replaces the parent (which is more similar to the selected child) with the child if the child's fitness is better than that of both parents,
7. repeats steps 3–6 until a convergence criterion is met.

Each GA agent

1. is waiting for an individual from the GA controller,
2. evaluates the received individual,
3. returns the evaluation result to the GA controller,
4. repeats steps 1–3.

Figure 3.8 shows the system entity structure [10] of the GA controlled high performance simulation environment. This environment can be implemented in two ways depending on available resources and the computational complexity of the simulation. The DAGA based implementation performs better than the AGA as the number of computing resources (processing elements) becomes larger and the computational complexity of simulation becomes smaller, as shown in Table 3.1. Otherwise the AGA based implementation is preferred since its implementation is simpler and more flexible than that of the DAGA.

3.3 Experimental Results of the DAGA

The DAGA described in the previous section was tested compared to the original AGA using the following seven test problems:

- Problem 1 (DeJong function 1):

$$f(x_i|_{i=1,3}) = \sum_{i=1}^3 x_i^2, x_i \in [-5.12, 5.12]$$

- Problem 2 (DeJong function 3):

$$f(x_i|_{i=1,5}) = 30 + \sum_{i=1}^5 [x_i], x_i \in [-5.12, 5.12]$$

- Problem 3 (DeJong function 5):

$$f(x_i|_{i=1,2}) = \sum_{j=1}^{25} (j + \sum_{i=1}^2 (x_i - a_{ij})^6), x_i \in [-65.536, 65.536]$$

- Problem 4 (Rastrigin function):

$$f(x_i|_{i=1,20}) = 200 + \sum_{i=1}^{20} x_i^2 - 10 \cos(2\pi x_i), x_i \in [-5.12, 5.12]$$

- Problem 5: order-5 deceptive problems
- Problem 6: Holland's revised Royal Road function
- Problem 7: Coloring mesh graphs

Problem 1–3 are borrowed from DeJong's suite [31]. DeJong function 1 is a unimodal function known to be easy for GAs. DeJong function 3 is a discontinuous step

ladder function. DeJong function 5 has several local minima. Rastrigin function is a very difficult problem for GAs because of the large search space (2^{200}) and large number of local minima [21]. Problem 5 is a class of order-five deceptive function, which is developed to deceive GAs [32]. We concatenated ten, twenty, thirty and forty size-five subfunctions together to form 50, 100, 150 and 200-bit problems. The order-five subfunction used here is a fully deceptive trap function of unitation [32] with value 0.58 at $u = 0$ ones, value 0.0 at $u = 4$ ones, and value 1.0 at $u = 5$ ones [33]. These deceptive functions are difficult problems. For example, the search space size of 200-bit problem is 2^{200} and it has 2^{40} optima, of which only one is global.

The Royal Road functions introduced in [34] were designed as functions that would be simple for GAs to optimize, but difficult for a class of hillclimbers. However, Holland recently revised the Royal Road functions since one form of hillclimbing outperformed GAs on this problem [35]. The revised Royal Road functions were designed to create insurmountable difficulties for a wider class of hillclimber, and yet can be optimized by GAs.

Problem 7 is a simple graph coloring problem. The problem is to color the $N \times N$ mesh graph with wraparound connections using two colors so that each vertex has neighbors with a different color. For an even number N , we have two solutions as shown in Figure 3.9. This problem has a large search space ($2^{N \times N}$) that is easily expandable.

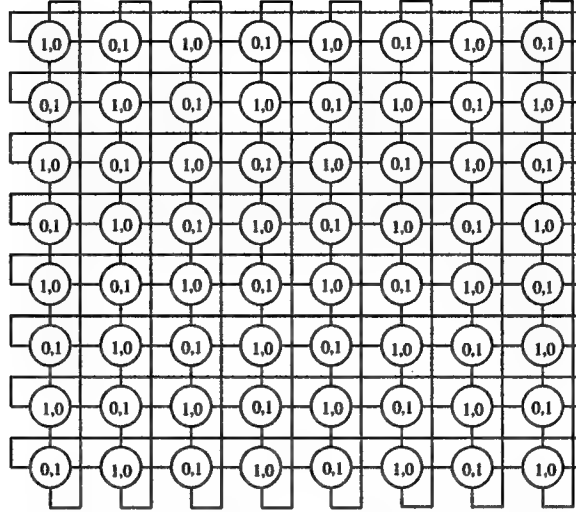


Figure 3.9: 8×8 mesh graph with wraparound connections (the numbers inside circles show two solutions for the coloring problem).

For all experiments we adjust the parameters of GAs, such as population size, crossover and mutation probability, to get the best performance based on the number of evaluations.

3.3.1 Results for DeJong's Suite

Function	F1		F2		F3	
Algorithm	evals	std	evals	std	evals	std
AGA	1,020	496	1,448	1,608	1,607	1,549
DAGA	2,766	2,000	3,250	1,465	1,484	793

Table 3.2: Performance of the AGA and DAGA on DeJong's test suite (evals: average number of evaluations, std: standard deviation).

We run the AGA and DAGA for 30 runs on DeJong's functions. We execute until the global optimum is found in all 30 runs and report the average number of evaluations and the standard deviation. As shown in Table 3.2, the performance

of the AGA is better than that of the DAGA on F1 and F3 which have only one global optimum. But the DAGA outperforms the AGA on F5 which has several local minima.

3.3.2 Results for Rastrigin function

Algorithm	solved	avg	avg. eval	total evals	total time(sec)	time/eval(μ sec)
AGA	5	0.9	390,356	11,710,680	18,540	1,583
DAGA	24	0.1	338,336	10,150,080	304	30

Table 3.3: Performance of the AGA and DAGA on Problem 4 on the CM-5 with 400 nodes (solved: number of runs solved, avg: average best of 30 runs after 400,000 evaluations, avg. eval: average number of evaluations, total evals: total number of evaluations, total time: total execution time in seconds, time/eval: time taken per one evaluation in microseconds).

For this problem, we run the AGA and DAGA for 30 runs with 400,000 evaluations and report the number of runs in which the global optimum is found with the average fitness of the best individuals at the end of each run. We set the population size as 400 to compare the results with those found in the literature [21]. As shown in Table 3.3, the performance of the DAGA for this problem is much better than that of the AGA. Compared to the results of the experiment done by Gordon [21], the DAGA outperforms the best one (Cellular GA) found in the literature on this problem.

We also measured the execution times of each GA (shown in the last column in Table 3.3). The DAGA is about 50 times faster than the AGA on CM-5 with 400 nodes.

3.3.3 Results for Problem 5, 6 and 7

Algorithm	avg. eval	std	problem size
AGA	51,053	6,582	50
AGA	124,600	14,161	100
AGA	240,429	35,028	150
AGA	408,822	60,233	200
DAGA	45,372	6,439	50
DAGA	101,806	8,770	100
DAGA	166,411	16,186	150
DAGA	252,443	30,496	200

Table 3.4: Performance of the AGA and DAGA on Problem 5 (solved: number of runs solved, avg. eval: average number of evaluations, std: standard deviation, problem size: problem size in bits).

Algorithm	solved	avg. evals	std
AGA	21	6,453,834	4,479,478
DAGA	50	107,718	47,122

Table 3.5: Performance of the AGA and DAGA on Problem 6 (avg. eval: average number of evaluations, std: standard deviation).

Algorithm	solved	avg. eval	std	problem size
AGA	50	1,111	38	4×4
AGA	50	9,344	1,832	8×8
AGA	43	2,018,566	3,395,681	16×16
DAGA	50	354	231	4×450
DAGA	50	10,407	7,360	8×8
DAGA	50	501,824	254,129	16×16

Table 3.6: Performance of the AGA and DAGA on Problem 7 (solved: number of solved, avg. eval: average number of evaluations, std: standard deviation, problem size: the size of mesh graph).

The tables 3.4, 3.5 and 3.6 show the performance of the DAGA compared to the AGA on Problem 5, 6 and 7, respectively. The results show that the DAGA reliably solve all three problems with a large search space.

The DAGA tested in this section outperforms (or is comparable to) those GAs found in the literature[21], [36] and [34].

3.4 Application Example: Design of a Fuzzy Controller for the Inverted Pendulum

In this section, we demonstrate the effectiveness of the DAGA by designing a fuzzy controller for a benchmark system in intelligent control — the Inverted Pendulum.

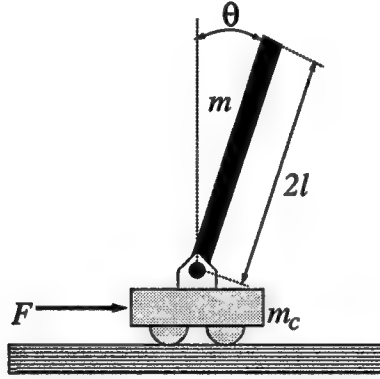


Figure 3.10: The Inverted Pendulum

Figure 3.10 shows the schematic diagram of an inverted pendulum system. Our control goal is to balance the rigid pole by exerting appropriate force F to the cart. Let $x_1(t) = \theta(t)$ and $x_2(t) = \dot{\theta}(t)$, then this system can be defined by the following differential equations [37] :

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{g \sin(x_1) + \cos(x_1) \left(\frac{-F - m l x_2^2 \sin(x_1)}{m_c + m} \right)}{l \left(\frac{4}{3} - \frac{m \cos^2(x_1)}{m_c + m} \right)} \\ &= H_2(x_1, x_2, F) \end{aligned}$$

where g is 9.8 meter/sec^2 , m_c (mass of cart) is 1.0 kg , m (mass of pole) is 0.1 kg , l (half length of pole) is 0.5 meter , and F is the applied force in newton. Using a two-step forward Euler integration we can approximate its states at time $t + h$:

$$x_1(t + 0.5h) = 0.5hx_2(t) + x_1(t)$$

$$x_2(t + 0.5h) = 0.5hH_2(x_1(t), x_2(t), F) + x_2(t)$$

$$x_1(t + h) = 0.5hx_2(t + 0.5h) + x_1(t + 0.5h)$$

$$x_2(t + h) = 0.5hH_2(x_1(t + 0.5h), x_2(t + 0.5h), F) + x_2(t + 0.5h)$$

where $h = 0.01 \text{ sec}$.

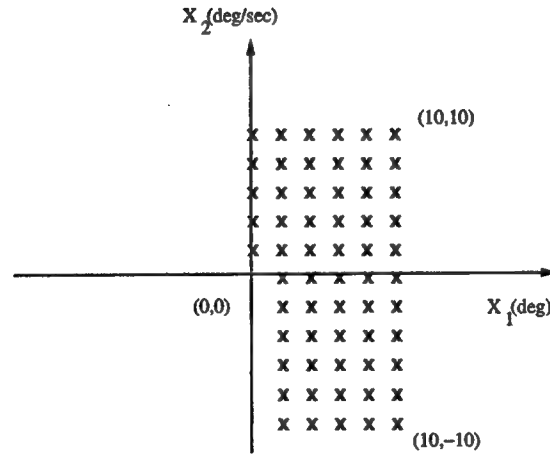


Figure 3.11: Initial conditions

We define 3 fuzzy regions(NE,ZE,PO) for each input(x_1, x_2) and output(F), resulting in 9 membership functions and 9 control rules. By symmetry only half the rules need be found.

Each membership function has a bell shape and the defuzzification layer uses the weighted average method as described in Appendix A. We assume that the inverted pendulum is expected to start from an initial point nearby the origin in the state space, therefore we optimize the membership functions and control rules using the GA optimizer for 60 initial conditions as shown in 3.11.

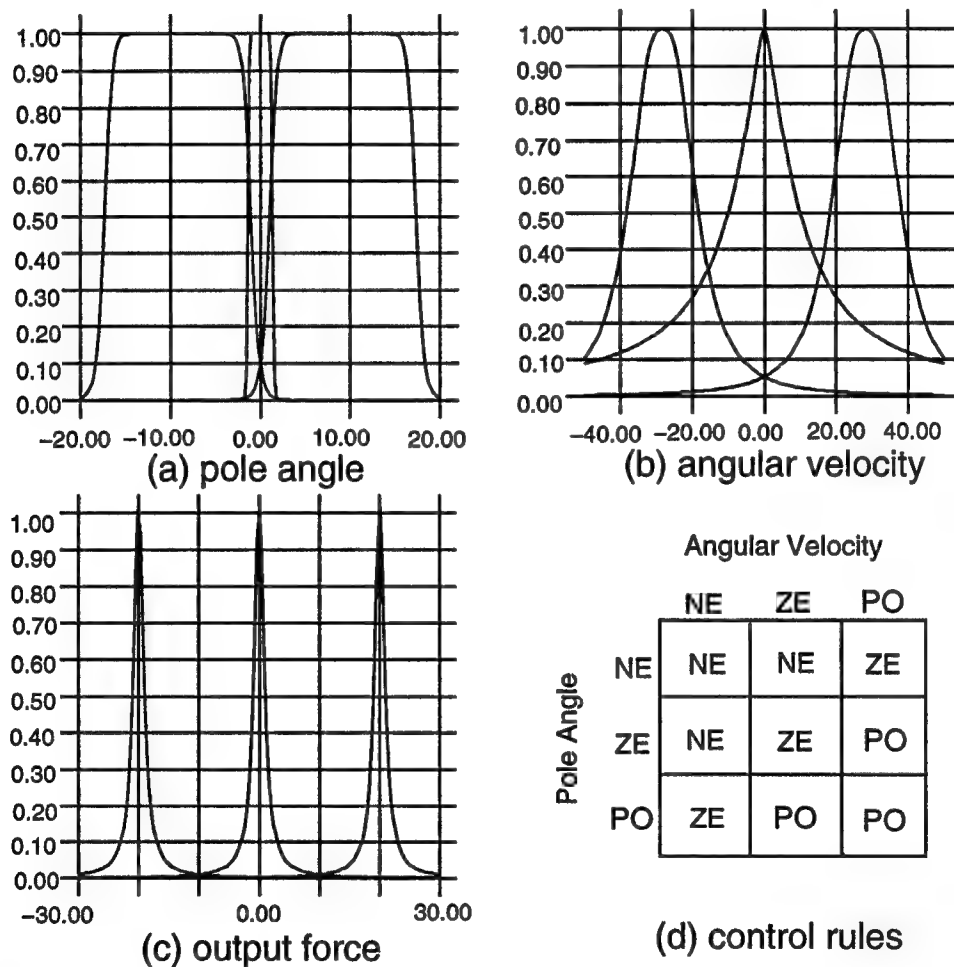


Figure 3.12: Optimized membership functions and control rules.

Figure 3.12 shows final membership functions and rules obtained, and Figure 3.13 exhibits how the fuzzy controller designed by the GA optimizer can balance the pole

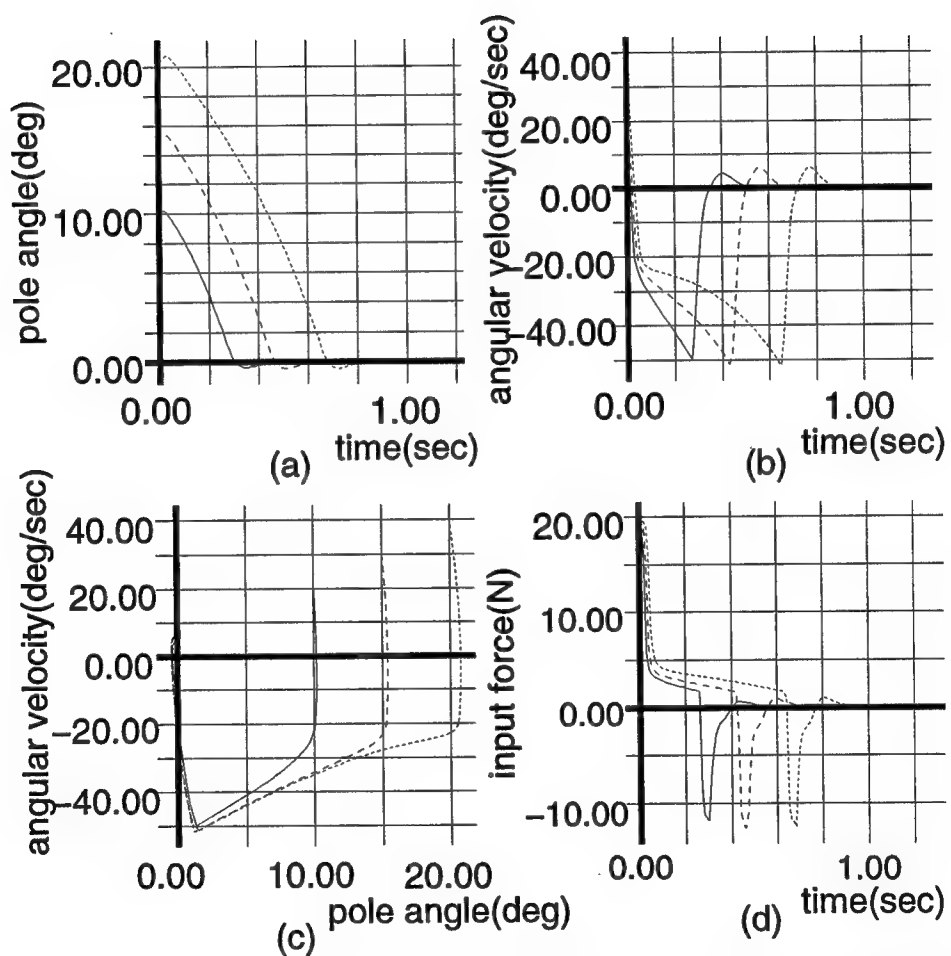


Figure 3.13: (a) Pole angle, (b) Pole angular velocity, (c) Phase plane trajectories ending at (0,0), and (d) Input force. (Solid, dashed, and dotted curves correspond to initial conditions (10,20), (15,30), and (20,40), respectively).

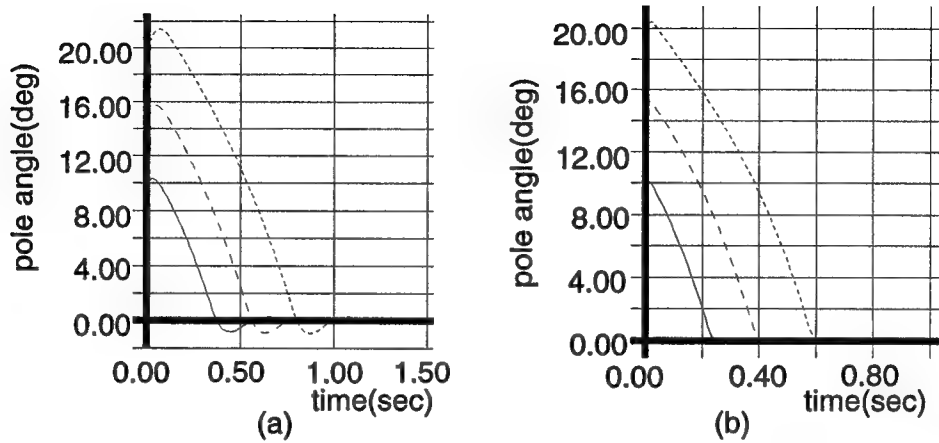


Figure 3.14: (a) Pole length = 2.0 m, (b) Pole length = 0.5 m, (Solid, dashed, and dotted curves correspond to initial conditions (10,20), (15,30), and (20,40), respectively).

from 3 different initial conditions. Although this fuzzy controller is designed for the initial conditions shown in Figure 3.11, it can also balance for initial conditions outside that region. Furthermore, this fuzzy controller can balance poles of different lengths as shown in Figure 3.14. Our results significantly improve upon those of [37].

We have demonstrated the use of the DAGA in Computer-Aided System Design. The DAGA optimizer for the fuzzy controller affords more reliability in global optimization than does an adaptive neural net approach [38]. We showed how the DAGA optimizer helps design a control system even for complex operational specifications.

With the CM-5 supercomputer used in these studies, typical optimization runs are completed within several minutes. Even with a single workstation, it can be completed within a day. Thus the use of the DAGA in control system design is feasible right now.

3.5 Implementation of GA C++

In this section we describe the implementation of the AGA and DAGA on different platforms in the object oriented fashion using C++. The software package that we implemented, called the GA C++, supports the AGA and DAGA on the CM-5 super computer and various workstations including easy interface to any simulators defined by the users.

As shown in Figure 3.15, the AGA controller resides on one workstation or one node on CM-5 and simulators are distributed to other workstations or nodes. The DAGA is implemented only on CM-5. The DAGA agent and simulator is running on each node on CM-5.

Figure 3.16 shows the classes of GA C++. The class GA has basic methods for GA operators such as crossover, mutation and replace. Under the class GA, there are three subclasses, AGA-CM5, AGA-PVM and DAGA Agent. These three subclasses inherit basic methods from the GA class and control simulators distributed to other nodes or workstations with their own start methods as explained in Section 3.2. The difference between AGA-CM5 and AGA-PVM is only in communication methods between the AGA controller and simulators. AGA-CM5 is an implementation for the CM-5 and uses built-in CM-5 CMMD communication libraries. AGA-PVM is an implementation for workstations using PVM.

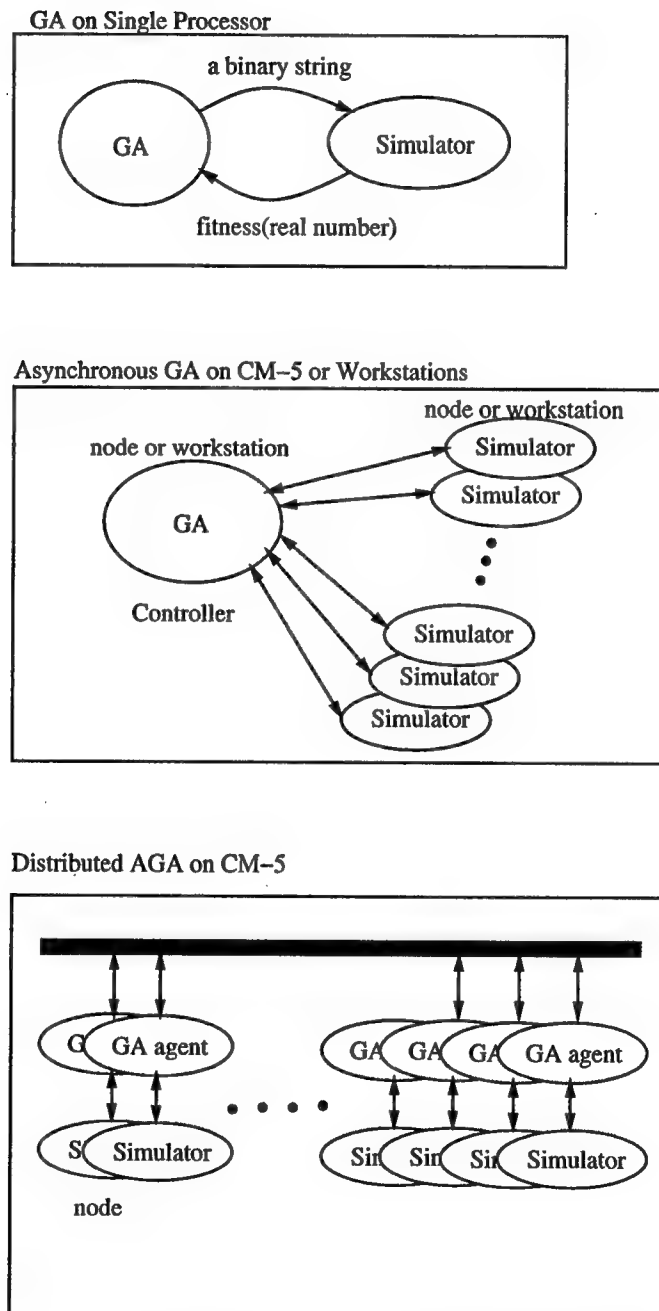


Figure 3.15: Implementation of GA C++.

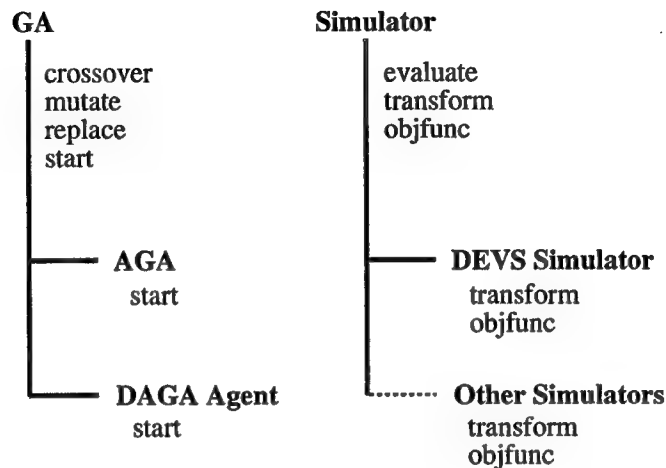


Figure 3.16: GA C++ and Simulator classes.

The class Simulator is an abstract class for interface between GA C++ and user-defined simulators. Simulator class provides three basic methods, *evaluate*, *transform* and *objfunc*. Using these three methods a user easily combines his/her own simulation and GA C++.

Figure 3.17 shows an example of the use of GA C++ on a single workstation environment. In User.main program an object called *sim* is created from the User-defined-simulator class (which should inherit Simulator class and whose two methods, *transform* and *objfunc* are defined by the user) and passed to *ga* object created from GA class. Every cycle in GA.start we call *evaluate* method with an argument of newly created child from parents. Simulator.evaluate transforms childs chromosome (defined as a character string inside GA C++) to a set of parameters using *transform* method defined by the user and obtains fitness of child by calling *objfunc* which can be any simulation or simple function evaluation defined by the user.

```
Program User.main
begin
    User-defined-simulator * sim = User-defined-simulator();
    GA * ga = GA(sim, );
    ga->start();
end

Program GA.start
begin
    initialize;
    while(evalno++<maxeval) {
        select parents;
        crossover;
        mutate;
        sim->evaluate(individual * child);
        replace;
        if (optimum found) break;
    }
end

Program Simulator.evaluate(individual * ind)
begin
    double * param = transform(ind->chromosome);
    ind->fitness = objfunc(param);
end
```

Figure 3.17: Code example for GA C++ and Simulator.

As shown in Figure 3.16, we have a predefined subclass called DEVS Simulator which is a simulator based on the DEVS formalism. In the following chapters, We show some examples of simulation based optimization with GA C++ and DEVS C++ Simulator.

CHAPTER 4

DEVS Modelling Example: Watershed

An example of distributed watershed hydrology will illustrate modelling and simulation in the high performance simulation based optimization environment.

Rainfall runoff in a watershed is a complex process. Many factors influence this process, including the conditions of the soil surface and its vegetative cover, the properties of the soil such as its porosity and hydraulic conductivity, and the current moisture content of the soil.

The complexity of watershed hydrology calls for powerful modelling methodologies able to handle spatial interaction over a heterogeneous landscape as well as temporal dynamics introduced by varying rainfall conditions. Geographic Information Systems (GIS) can provide the spatially referenced data necessary to represent topography, rainfall, and soil state distributions. Spatial dynamic models are needed to project such states forward in time. However conventional differential equation formulations entail an enormous computational burden that greatly limits their applicability. By combining GIS, for state characterization, and DEVS, for dynamic state projection, we derive an approach that can achieve realism within feasible computational constraints, albeit in high performance environments.

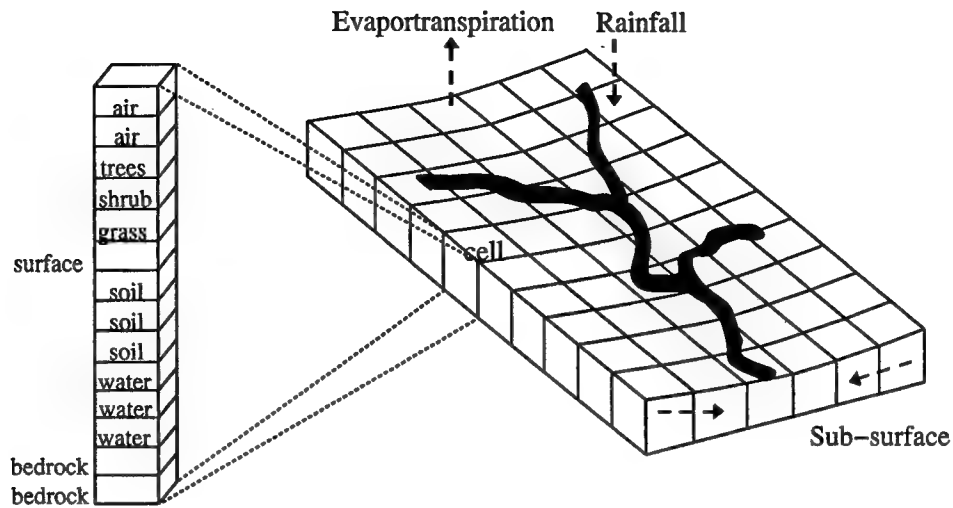


Figure 4.1: Grid based representation of a watershed.

Figure 4.1 shows a typical watershed, which consists of several vertical layers, such as air, surface water, subsurface soil, ground water and bedrock. We divide it into many small cells and develop a conceptual hydrology model for each cell that can be readily mapped into a DEVS component model. Then we define how the directions of water flow are coded in a grid space and how the varying influx rates in the discretized landscape are linked to create a coherent total runoff.

4.1 A Conceptual Hydrology Model for a Cell

As shown in Figure 4.2, we conceptually represent a cell with three vertically connected reservoirs. The rainfall input ($r(t)$) is partially intercepted by vegetation cover and the rest of it, the effective rainfall ($r_e(t)$), becomes the source of surface runoff and the infiltration. The surface reservoir receives the inputs, the effective rainfall (r_e) and inflow ($qi(t)$) from the neighbor cells, and generates the outputs, the

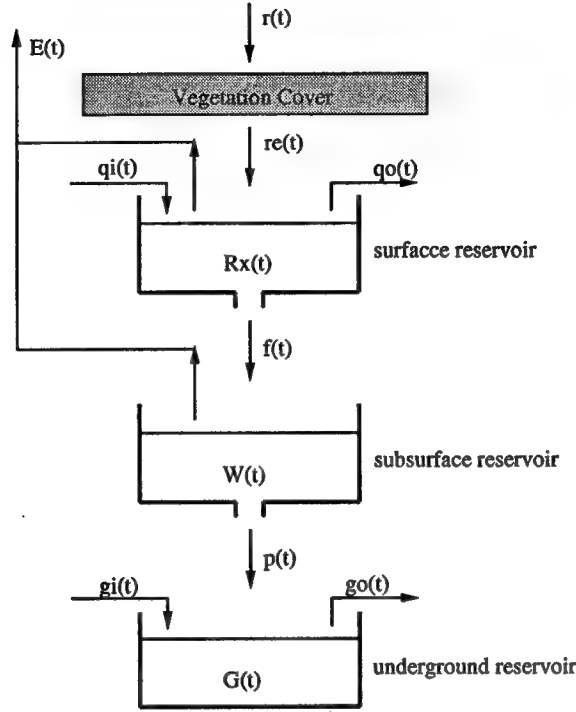


Figure 4.2: Conceptual hydrology model for a cell.

runoff ($qo(t)$) to neighbor cells and infiltration($f(t)$) to the subsurface reservoir. The underground reservoir works similar to the surface reservoir except for infiltration.

We define the water depth on a cell, the rainfall excess ($R_x(t)$), and the runoff ($qo(t)$) as follows:

$$R_x(t) = \int_0^t (r_e(t) + \sum_i qi_i(t) - f(t) - \sum_i qo_i(t))dt \quad (4.1)$$

$$qo_i(t) = \frac{C(S_i(t))^a (R_x(t))^b}{W_i} \quad (4.2)$$

where

$qi_i(t)$: inflow from i th neighbor cell,

$qo_i(t)$: runoff to i th neighbor cell,

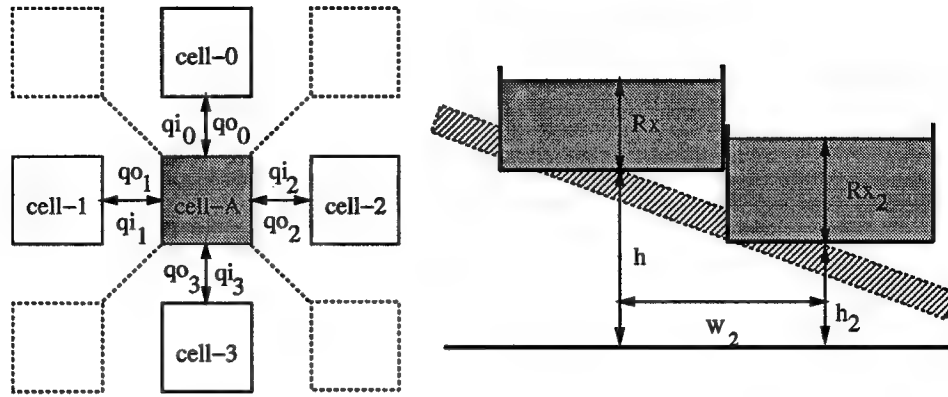


Figure 4.3: Connection of Cells

C : a parameter characterizing the surface roughness at the cell's location,

$S_i(t)$: slope to the i th neighbor cell,

a, b : some constants,

W_i : distance to the i th neighbor cell.

The slope S_i is computed by:

$$S_i(t) = \frac{h - h_i + P(R_x(t) - R_{xi}(t))}{W_i}, \quad (4.3)$$

where h = altitude of the cell, R_{xi} = rainfall excess of i th neighbor cell, h_i = altitude of i th neighbor cell, P = a constant (refer to Figure 4.3).

As shown in Figure 4.3, each cell can have at most eight neighboring cells. However, we may consider only four connections by ignoring diagonal neighbor cells or even one connection (the direction of maximum runoff) depending on communication overhead costs and required accuracy of simulation results. Some experimentation indicates that there is not much difference in flow patterns between 4 and 8 neighbors

models but that the "gradient" (maximum flow) generates a distinctly different, and less realistic looking behavior.

4.2 DEVS representation of Infiltration

In this section, we show how DEVS can represent the continuous infiltration process.

During a rainstorm, the rate of rainfall changes constantly. Partly because of limitations in measuring equipment, we commonly approximate this rate change with a finite number of relatively short pulses. Each pulse is assumed to have a constant rate, but the rate changes from pulse to pulse. This sequence of rainfall pulses is both temporally and spatially distributed.

Infiltration is the process by which portions of the rain that are not intercepted by plants or surface litter enter the soil. The infiltration rate is not constant. Its pattern responds to the variation in rainfall rates and to the accumulated infiltration amount. If the area of each cell in the grid based representation of watershed (Figure 4.1) is small enough, it can be considered as a point source. There are several mathematical models to compute infiltration for such a point source and most of them have the form of nonlinear differential equations which are generally solved by iteration methods such as the Newton-Raphson method. Discrete event simulation has been shown to afford many advantages such as flexibility and efficiency over continuous simulation for large scale landscape models [39].

Although the continuous system model described by a set of nonlinear differential equations can be directly converted to a DEVS model without considering efficiency, we need a model that fully takes advantage of discrete event simulation [40].

We adopt the approach of [40] to develop a DEVS model for infiltration by abstracting a continuous model described by the Green-Ampt equation [41]. A fuzzy system is designed to solve the Green-Ampt equation for significant events of the infiltration process without using iteration methods in the proposed environment.

4.2.1 Green-Ampt Infiltration Model

The Green-Ampt equation has become widely used to compute infiltration in catchment-scale hydrologic models [42, 43]. In addition to the fact that the parameters in the equation have physical significance, experimental works have been completed or are underway to obtain values for the parameters based on soil texture and on the effects of management [44].

The rate form of the Green-Ampt equation for the one stage case of initially ponded condition is

$$f_c(t) = K_e \left(1 + \frac{\psi \theta_d}{F(t)} \right) \quad (4.4)$$

where $f_c(t)$ = infiltration capacity(L/T), K_e = effective saturated hydraulic conductivity(L/T), ψ = average capillary potential at the wetting front(L), θ_d = soil moisture deficit(L/L), and $F(t)$ = cumulative infiltrated depth(L) (note: L and T represent length and time for all variables). The soil moisture deficit can be computed

$$\theta_d = \theta_s - \theta_i = \eta(S_{max} - S_i) \quad (4.5)$$

where θ_s = volumetric water content at saturation(L/L), θ_i = initial volumetric water content(L/L), η = soil porosity, and S_{max} and S_i are maximum and initial values of relative saturation.

Recognizing that $f_c(t) = \frac{dF(t)}{dt}$, we integrate this relation to obtain

$$F(t) = K_e t + \psi \theta_d \ln[1 + \frac{F(t)}{\psi \theta_d}] \quad (4.6)$$

Equation 4.6 is normally solved numerically for successive increments of time using the Newton-Raphson iteration method.

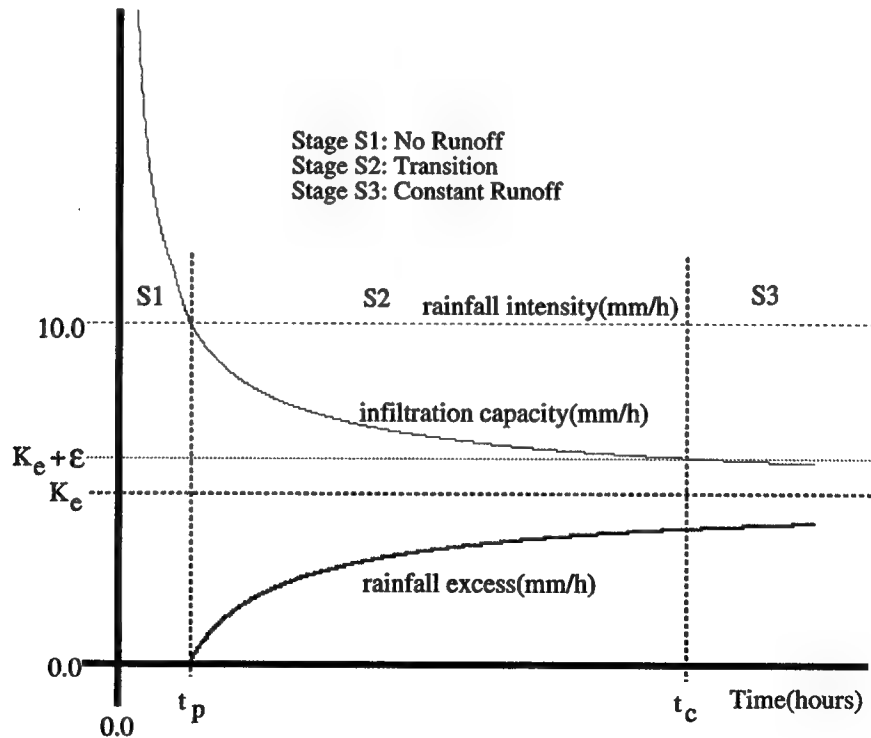


Figure 4.4: Green-Ampt infiltration model.

Figure 4.4 shows the infiltration capacity and the rainfall excess during a constant rainfall. There is no rainfall excess until the rainfall intensity becomes larger than the infiltration capacity. The rainfall intensity is larger than the infiltration capacity after the time to ponding (t_p) where the rainfall intensity equals to the infiltration capacity. Equation 4.4 shows that the infiltration capacity f_c asymptotically approaches to K_e , and the rainfall excess can be considered as a constant after the time to constant runoff (t_c) where f_c becomes $K_e + \epsilon$ for small ϵ . We can divide the infiltration process into three stages, a stage without runoff, a stage with transitional runoff and a stage with constant runoff using t_p and t_c as in Figure 4.4.

4.2.2 DEVS Model for Infiltration

The infiltration process in Section 4.2.1 can be described by the DEVS formalism in an efficient way if the following can be calculated:

1. The time to ponding ($t_{to-ponding}$) from any time t where the rainfall intensity changes in stage S1. This time is a function of the rainfall intensity and the cumulative infiltrated depth at t .
2. The time to constant runoff from t_p ($t_{to-const} = t_c - t_p$) where the rainfall excess can be considered as a constant. This time is only a function of the cumulative infiltrated depth at t_p .
3. The cumulative infiltrated depth $F(t_c)$.

A DEVS model M for infiltration can be defined as

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

where $X = \{r_{in} \mid r_{in} = \text{input rainfall intensity (L/T)}\}$, $Y = \{y \mid y = \text{runoff (L/T)}\}$, $S = \{s \mid s = (\text{phase}, \text{sigma}, r_{cur}, r_e, F, T_{to-const})\}$, r_{cur} = current rainfall intensity (L/T), r_e = rainfall excess (L), F = cumulative infiltrated depth (L), and $T_{to-const}$ = remaining time to constant runoff.

In Section 4.2.1 we divided the infiltration process into three stages—*NoRunoff* (stage S1), *Transition* (stage S2), and *ConstantRunoff* (stage S3). We define *phase* as one of *NoRunoff*, *Transition*, and *ConstantRunoff* for each stage S1, S2, and S3, respectively. In addition to these three phases we need two more phases, *Transition^o* and *ConstantRunoff^o*, to generate the output when the rainfall intensity changes during stages S2 and S3.

The internal transition function δ_{int} , the external transition function δ_{ext} , and the time advance function ta for the DEVS model M are shown below.

- The internal transition function $\delta_{int}(s) = s'$, where $s' = (\text{phase}', \text{sigma}', r_{cur}', r_e', F', T_{to-const}')$

When *phase* = *NoRunoff*

$$F' = F + r_{cur} * \text{sigma}$$

$$r_e' = \frac{r_{cur} * t_{to-const}(F') - (F(t_e) - F')}{t_{to-const}(F')}$$

$$phase' = Transition$$

$$sigma' = ta(s) \quad (Variables \text{ not shown are unchanged.})$$

When $phase = Transition$

$$F' = F(t_c)$$

$$r'_e = r_{cur} - K_e$$

$$phase' = ConstantRunoff$$

$$sigma' = ta(s)$$

When $phase = Transition^o$

$$phase' = Transition$$

$$sigma' = T_{to-const}$$

When $phase = ConstantRunoff^o$

$$phase' = ConstantRunoff$$

$$sigma' = ta(s)$$

- The external transition function $\delta_{ext}(s, e, x) = s'$, where $s' = (phase', sigma',$

$$r'_{cur}, r'_e, F', T'_{to-const})$$

When $phase = NoRunoff$

$$F' = F + r_{cur} * e$$

$$r'_{cur} = r_{in}$$

$$sigma' = ta(s)$$

When $phase = Transition$

$$T'_{to-const} = sigma - e$$

$$r'_e = r_e - (r_{cur} - r_{in})$$

$$r'_{cur} = r_{in}$$

$$phase' = Transition^o$$

$$sigma' = 0$$

When $phase = ConstantRunoff$

$$F' = F + K_e * e$$

$$r'_{cur} = r_{in}$$

$$r'_e = r'_{cur} - K_e$$

$$phase' = ConstantRunoff^o$$

$$sigma' = 0$$

- The time advance function $ta(s)$:

When $phase = NoRunoff$

$$ta(s) = t_{to-ponding}(r_{cur}, F)$$

When $phase = Transition$

$$ta(s) = t_{to-const}(F)$$

When $phase = ConstantRunoff$

$$ta(s) = \infty$$

- The output function $\lambda(s)$:

At the end of *phase = NoRunoff*

$$\lambda(s) = \frac{r_{cur} * t_{to-const}(F) - (F(t_c) - F)}{t_{to-const}(F)}$$

At the end of *phase = Transition^o*

$$\lambda(s) = r_e$$

At the end of *phase = Transition*

$$\lambda(s) = r_{cur} - K_e$$

At the end of *phase = ConstantRunoff^o*

$$\lambda(s) = r_e$$

The operation of the model M is as follows:

1. At time $t = 0$, the *phase* is *NoRunoff* and the next event is scheduled as $\sigma = t_{to-ponding}(r, F)$, where r is the rainfall intensity and F is the cumulative infiltrated depth at $t = 0$.
2. If the rainfall intensity changes at time t_1 during stage S1, the next event is rescheduled as $\sigma = t_{to-ponding}(r, F)$.
3. Note that between t_p and t_c the rainfall excess and, therefore the output runoff, varies. A DEVS model, however, can only approximate this curve by finite number of outputs. If the rainfall intensity doesn't change during stage S2, the output at t_p is to represent the rainfall excess between t_p and t_c . Conservation of mass requires that $r_e \times (t_c - t_p) = \text{total runoff}$. Therefore, at time $t = t_p$, the

- rainfall excess is approximated as $r_e = \frac{r * t_{to-const}(F_p) - (F(t_c) - F_p)}{t_{to-const}(F_p)}$, where r is the rainfall intensity, $t_{to-const}(F_p) = t_c - t_p$, F_p is the cumulative infiltrated depth at t_p , and $F(t_c)$ is the cumulative infiltrated depth at time t_c . The next internal event is scheduled as $sigma = t_{to-const}(F_p)$.
4. Consider a rainfall intensity change at $t_1 (t_p \leq t_1 \leq t_c)$. We update $sigma = sigma - e$, where e is time elapsed since the last internal or external model event. The rainfall excess r'_e is recalculated as $r'_e = r_e - (r_{cur} - r_{in})$ (recall that r_{cur} = current rainfall intensity and r_{in} = new input rainfall intensity). Note that the rainfall intensity change only affects the rainfall excess but not the infiltration process [41].
 5. At time $t = t_c$, the rainfall excess is calculated as $r_{cur} - K_e$. The next event is scheduled as $sigma = \infty$, i.e., the model will remain passive unless activated by an external event.
 6. The rainfall intensity change at time t during stage S3 recalculates the rainfall excess r'_e as $r'_e = r - K_e$, where r is the rainfall intensity at time t .

To realize the above DEVS model we need to represent the Green-Ampt solution for $t_{to-ponding}$, $t_{to-const}$, and $F(t_c)$. Although the DEVS model for infiltration approximates the rainfall excess for the transitional stage as finite number of outputs, this is not a major source of error for a long term simulation of a large watershed represented by a grid system of small cells — the intended application.

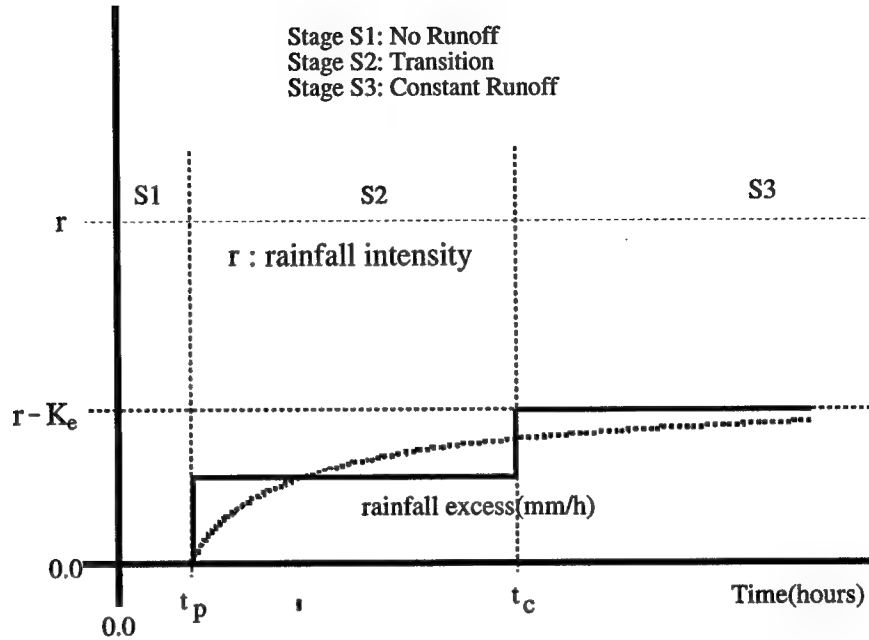


Figure 4.5: DEVS approximation model behavior (Solid and dashed curve represent the outputs of the DEVS and continuous model, respectively).

Figure 4.5 shows the behavior of the DEVS model for infiltration during a constant rainfall compared to that of the continuous system model. For constant rainfall input, the DEVS model approximates the continuous curve of rainfall excess in two steps. Since only two computations are needed, simulation of the DEVS model is more efficient than that of the continuous model which may need thousands of small steps.

4.2.3 Implementing a DEVS Model using a Fuzzy System

In Section 4.2.2, we presented a DEVS model for infiltration, but we need solve the Green-Ampt equation for three unknowns, $t_{to-ponding}$, $t_{to-const}$, and $F(t_c)$, to implement it. We can analytically solve the Green-Ampt equation for $t_{to-ponding}$ in the case that the duration of a rainfall event is divided into many short periods in

such a way that within each period the rainfall intensity is essentially constant [45]. Assuming that the rainfall intensity changes from r_{prev} to r_{cur} at time t for $t \leq t_p$, we can calculate the time to ponding $t_{to-ponding}$ since we know that the infiltration capacity ($f_c(t_p)$) and the cumulative infiltrated depth (F_p) at t_p should be r_{cur} and $r_{prev}t + r_{cur}t_{to-ponding}$, respectively. If the rainfall intensity changes more than once before time t , then F_p should be $F(t) + r_{cur} \times t_{to-ponding}$, where $F(t)$ is the cumulative infiltrated depth at time t . Using Equation 4.4, the time to ponding from any time t can be calculated as

$$t_{to-ponding}(r_{cur}, F(t)) = \frac{K_e F(t) + K_e \psi \theta_d - r_{cur} F(t)}{r_{cur}(r_{cur} - K_e)} \quad (4.7)$$

and $t_{to-ponding}$ is a function of the rainfall intensity r_{cur} and the cumulative infiltrated depth F at time t .

We defined the time t_c as the instant when the infiltration capacity $f(t_c)$ becomes $K_e + \epsilon$, and from Equation 4.4, the cumulative infiltrated depth $F(t_c)$ is

$$\frac{K_e \psi \theta_d}{\epsilon} \quad (4.8)$$

The Green-Ampt equation allows us to solve it for the time to ponding and the cumulative infiltrated depth $F(t_c)$ analytically, but it doesn't allow us to solve it for $t_{to-const}$ which is a function of F_p . There are several ways to solve this problem. One possible solution is to use approximation forms of the Green-Ampt equation which allow analytical solution for $t_{to-const}(F_p)$ [46]. The second one is to maintain a lookup table which contains all solutions for every possible input event. However this scheme

requires a lot of memory. Another way is to maintain the approximate solutions using different types of computational structure such as neural networks or fuzzy systems. In this paper we use a fuzzy system described in Appendix A to hold the solutions of $t_{to-const}(F_p)$ for the whole range of F_p .

The range of F_p can be calculated from Equation 4.4 for a given soil, a range of input rainfall intensity, and an infiltration capacity at t_c . At t_p , the rainfall intensity r is f_c and the cumulative infiltrated depth F is the ponding depth F_p . From Equation 4.4,

$$r = K_e(1 + \frac{\psi\theta_d}{F_p})$$

Thus,

$$F_{pmin} = \frac{K_e\psi\theta_d}{r_{max} - K_e} \quad (4.9)$$

where F_{pmin} is the minimum ponding depth and r_{max} is the maximum rainfall intensity. Let the infiltration capacity and the cumulative infiltrated depth at t_c be $f_c(t_c)$ and $F(t_c)$, respectively, then the maximum ponding depth F_{pmax} is $\frac{K_e\psi\theta_d}{f_c(t_c) - K_e}$ since $F_p \leq F(t_c)$. If we define $f_c(t_c)$ as $K_e + \epsilon$, then

$$F_{pmax} = \frac{K_e\psi\theta_d}{\epsilon} \quad (4.10)$$

Given the range of F_p as in equations 4.9 and 4.10, we can solve the Green-Ampt equation for $t_{to-const}$ for a given number of training input points and design a fuzzy system that approximates the solutions using the AGA optimizer.

4.2.4 Experimental Results

Green-Ampt parameter values for a silt soil are $K_e = 5.0$ (mm/h), $\psi = 190.0$ (mm), and $\eta = 0.42$ [44]. Let the maximum rainfall intensity r_{max} be 200.0 (mm/h), S_{max} be 1.0 and S_i be 0.5, then F_{pmin} is 1.02 (mm) from Equation 4.9. We chose ϵ to be $K_e \times 0.2$ to keep t_c close to 1 day, and F_{pmax} is calculated as 199.50 (mm) using Equation 4.10. The possible range for the output ($t_{to-const}$) can be obtained by solving the Green-Ampt equation using an iteration method. As shown in Figure 4.6, $t_{to-const}$ has a value between 0.0 (hours) and 32.0 (hours).

It may happen that a better fuzzy approximation is obtained by locating the center of some membership functions outside the range calculated above. Therefore, we extended the search space by 60% and defined three fuzzy regions for $-60.0 \leq F_p \leq 260$ and $-9.6 \leq t_{to-const} \leq 41.6$. We then optimized the membership functions and rules at the same time using the DAGA optimizer described in Chapter 3.

Figure 4.6 shows the solution for $t_{to-const}$ by the fuzzy system compared to the solution by the Newton-Raphson method, which is the target of the fuzzy system, and the solution by the two-term Taylor series approximation.

The experimental results show that the fuzzy system can solve the Green-Ampt equation for $t_{to-const}$ within 0.3 hours maximum error. The fuzzy system approximates $t_{to-const}$ better than the two-term Taylor series approximation of the Green-Ampt equation which recently appeared in the literature [46]. The latter suffers 3.5 hours maximum error as shown in Figure 4.6. The fuzzy system was trained using

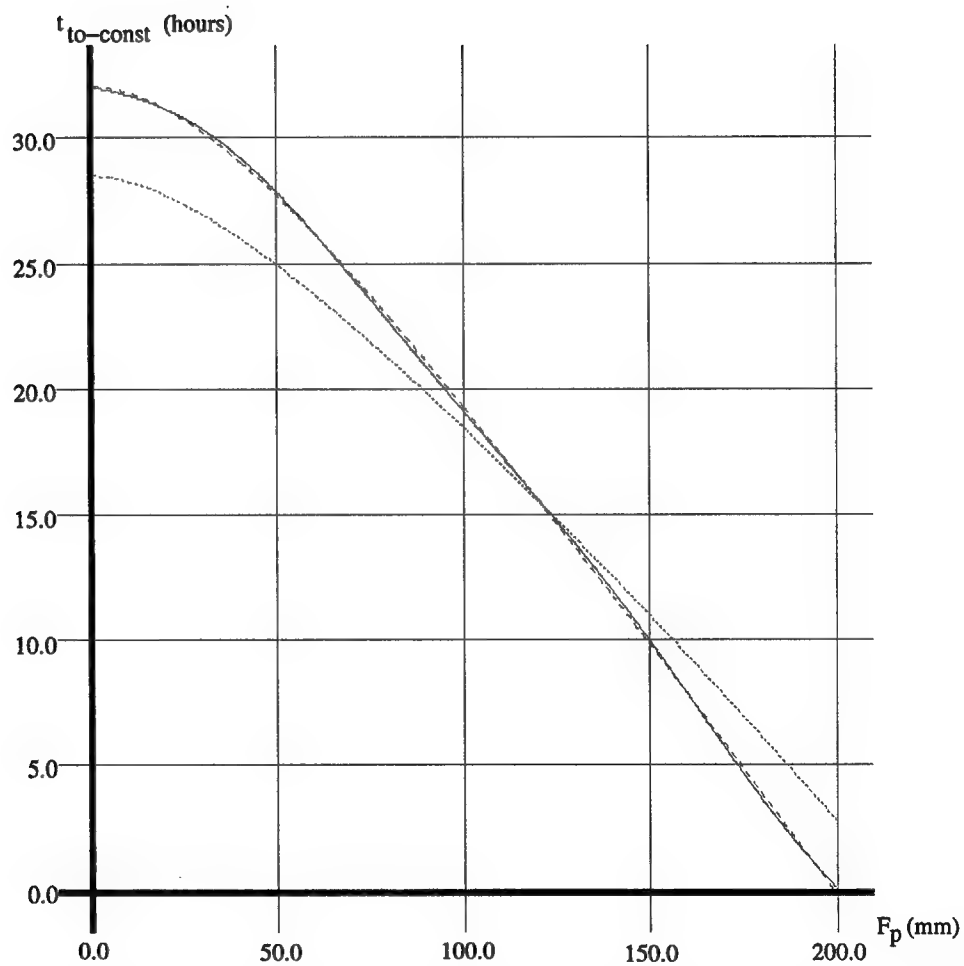


Figure 4.6: The time to constant runoff $t_{to-const}$ for the silt soil (Dashed, solid and dotted curves are obtained for the fuzzy system, Newton-Raphson method and two-term Taylor series approximation, respectively.).

only 100 data points while the curve shown in Figure 4.6 represents the output for 1,000 input points. The membership functions of the fuzzy approximation are shown in Figure 4.7 and the rules are as follows:

- Rule 1: If F_p is Small, $t_{to-const}$ is 35.2 hours.
- Rule 2: If F_p is Medium, $t_{to-const}$ is 31.7 hours.
- Rule 3: If F_p is Large, $t_{to-const}$ is -7.2 hours.

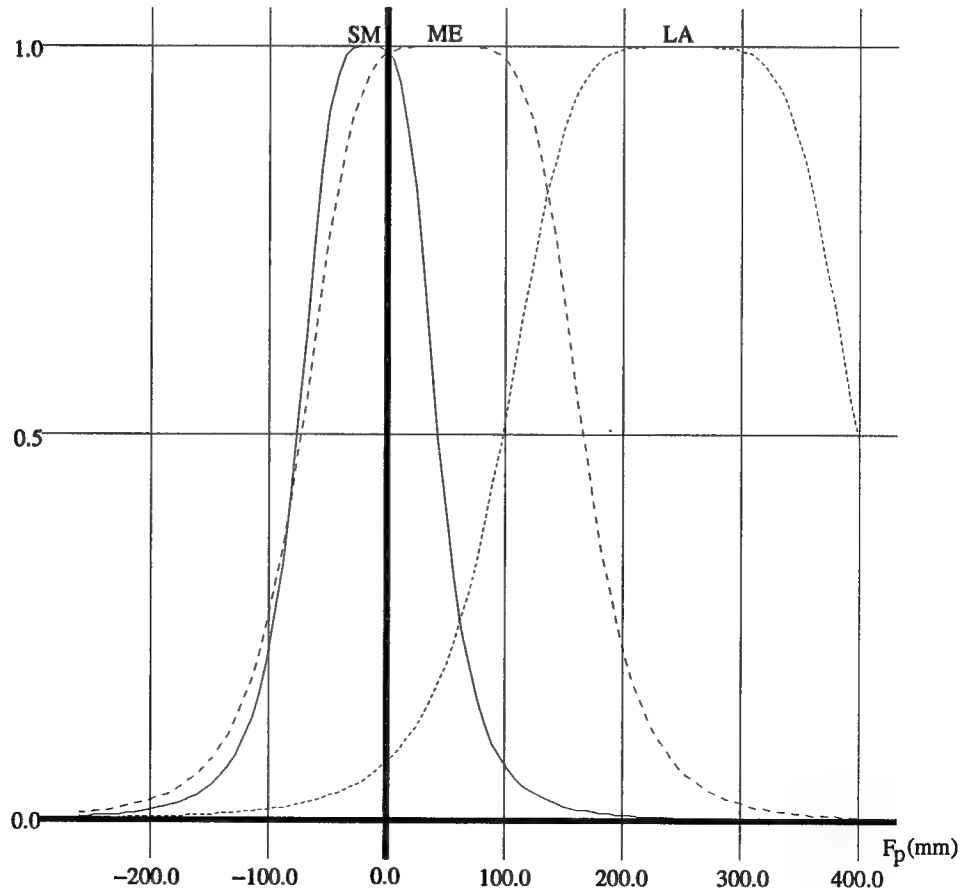


Figure 4.7: The membership functions of the fuzzy system (SM, ME, LA stand for small, medium and large, respectively.).

We need about 1,000,000 evaluations to optimize the fuzzy system. Execution times for this task for various processor sets on the CM-5 massively parallel computer are shown in Table 4.1. While it takes about 7 hours to complete an optimization run using 1 processor, this time is reduced to approximately 1 minute using 512 processors. The approximate speedup of 350 times is certainly significant in establishing the practicality of the approach.

# of nodes	1	32	64	128	256	512
execution time (sec)	25,221	1,205	615	281	149	72

Table 4.1: The execution time to optimize the fuzzy system on the CM-5 (measured for 1,000,000 evaluations).

We have devised a DEVS model for infiltration described by the Green-Ampt equation and shown that this model can be realized using a fuzzy system approximation designed by GA optimization on a CM-5 supercomputer. The fuzzy system outperforms the two-term Taylor series approximation proposed in [46] on the same data set.

The approach using fuzzy approximation requires offline training using GAs. However it has an important benefit. Real world observed data, alone or combined with that generated by a mathematical model, can be used to train the fuzzy membership functions. In contrast, the Taylor series approximation requires a mathematically tractable model.

We have also shown that fuzzy systems can represent the time to constant runoff for any one type of soil. However, if this technique were to be used in a watershed

consisting of many different types of soil, one fuzzy approximation for each would be required. So it remains of interest to design a fuzzy system that can represent the time to constant runoff for different types of soil with just one set of membership functions and employing one or more additional dimensions to represent soil characteristics.

Writing the DEVS model forced us to consider several state-input conditions that are not typically considered in the hydrology literature. By forcing us to provide behaviors under these conditions, the DEVS abstraction methodology afforded new insights into underlying real processes. It also stimulated us to plan new experiments to fill in gaps in our understanding of basic hydrologic processes.

4.3 DEVS Representation of Runoff

Recall that for a set of component models, a coupled model can be created by specifying how the input and output ports of the components will be connected to each other and to the input and output ports of the coupled model. Due to closure under coupling [15], the new coupled model is itself a modular model and thus can be used as a component in a yet larger, higher level model.

For the simulation of water flow in a cellular space one can envision the placement of an atomic model at each cell location. Thus there is an array of spatially referenced models that form a coupled DEVS model that can be coupled to an experimental frame component. DEVS *atomic models* are stand-alone modular objects that contain state variables and parameters, internal transition, external transition, output and

time advance functions. Two state variables are standard: *phase* and *sigma*. In the absence of external events the model remains in the current state, including *phase* state variable, for the time given by *sigma* at which time an internal transition occurs. If an external event occurs, the external transition immediately places the model into a new state, depending on the input event, the current state, the time that has elapsed in that state. The new state may have a new value for *sigma* thus scheduling the next internal transition. Note that DEVS recognizes the crucial role that the elapsed time plays in the external transition computation. This enables DEVS to faithfully represent the behavior of continuous systems through discrete events [11, 47].

The differential equation system described in Section 4.1 can be formalized in an atomic model *cell*. One way, equivalent to the conventional numerical analysis approach, is to transform the continuous system into a discrete time approximation. That is, we set *sigma* of the *cell* to some constant *d*. Each *cell* updates its states and generates outputs to neighbor *cells* at every fixed time step. However, while straightforward, updating the states of every cell every time step imposes a heavy computational burden that may be far more than necessary as suggested in Section 2.1.

A more efficient and conceptually satisfying approach is to partition the state space into output equivalent blocks as shown in Figure 4.8. While its state trajectory remains in a block, each *cell*'s output remains constant. Internal events of each *cell* correspond to boundary crossings in the *cell*'s state space. Given a state on a

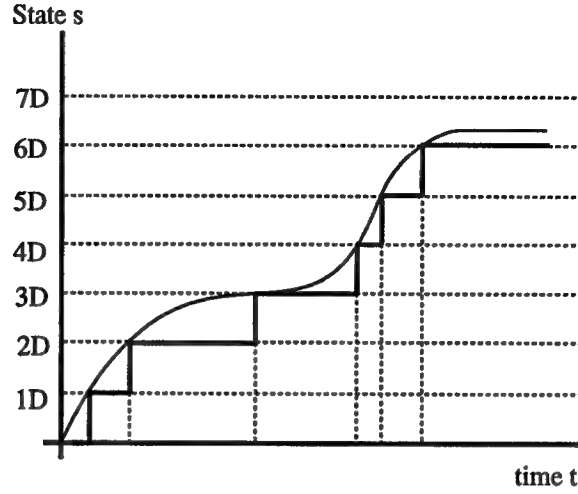


Figure 4.8: State space partitioning for DEVS representation.

boundary, each *cell* predicts the state that will be reached on the next boundary crossing and the time (*sigma*) to reach it. Due to the heterogeneity of soil conditions, slope, and input flux conditions each *cell* fills at a different rate and thus takes a different time to reach its next quantization level. Note that while it is in a quantum block, the cell's output fluxes to its neighbors are constant and all input fluxes are constant as well. Therefore, this enables us to compute when and where the next level crossing (increasing or decreasing) will occur.

For example, assume that in the *cell A* the rainfall excess (water depth) is $R_{x_A}(t)$ and rainfall excess rate (difference between input and output) is $r_{x_A}(t)$ at time t , then from Equation 4.2 and 4.3 the runoff to the i th neighboring cell ($q_{A_i}(t + \delta t)$) at time $t + \delta t$ is calculated by:

$$R_{x_A}(t + \delta t) = R_{x_A}(t) + r_{x_A}(t)\delta t$$

$$R_{x_i}(t + \delta t) = R_{x_i}(t) + r_{x_i}(t)\delta t$$

$$\begin{aligned}
S_i(t + \delta t) &= \frac{h_A - h_i + P(R_{x_A}(t + \delta t) - R_{x_i}(t + \delta t))}{W_i} \\
q_{A_i}(t + \delta t) &= \frac{C(S_i(t + \delta t))^a (R_{x_A}(t + \delta t))^b}{W_i},
\end{aligned} \tag{4.11}$$

where $R_{x_i}(t)$ and $r_{x_i}(t)$ are the i th neighboring cell's rainfall excess and rainfall excess rate at time t , respectively.

If the runoff of a cell to its neighbor is nD (for some integer n and quantum size D) at time t , then we can compute the time advance δt when the runoff q_{A_i} becomes $n(D - 1)$ or $n(D + 1)$ using Equation 4.11. The equation can analytically be solved for the cases when $P = 0.0$ or when $a = b = 1.0$. Otherwise it can be solved by iteration methods. Since there can be up to eight neighbor cells and each neighbor cell can be in different states, the times to next level crossing for each neighbor can be different. In this case we have to take the minimum of these times as *sigma*.

When a cell receives an external event from a neighbor, the message carries the latter's new output flux. The receiving cell's time and location of next boundary crossing may differ from that initially predicted. As indicated before, the DEVS formalism can handle this situation. Since elapsed time is known, the actual water level can be computed and *sigma* recalculated to represent reaching of the next quantum level at the new rate using Equation 4.11.

When the *quantized cell model* implemented in this way was tested, the results were disappointing. There was little if any reduction in computation time compared with discrete time models. One source of overhead that the quantized model entails,

not found in the discrete time model, is the extra calculation for *sigma*. However, analysis revealed that the main differentiating overhead was due to the structure of the DEVS-C++ simulator (described in Section 2.3). Consider the case where N cells, in different states, schedule their next events at different points on the real time axis. In this case, the DEVS-C++ simulator requires N iterations to execute all the events. This requires N times as many iterations than for a discrete time model in which the DEVS-C++ simulator updates all cells in one iteration. Note that the simulator can perform such a one iteration update since it implements the new parallel DEVS formalism where all the inputs and outputs of all simultaneously scheduled cells are properly managed. Thus the discrete time simulation is actually getting more of a boost than it would get in a conventional sequential cell scanning algorithm.

This analysis immediately suggested a remedy: squeeze events dispersed over the time axis that are "close enough" to each other into groups, that are executed in one iteration. (Note that this equivalencing is, in effect, an abstraction and is prone to introduce error into the representation.) To accomplish this effect, we quantize the time axis with a *time granule* of size d , in addition to quantizing the state space. The events between t and $t + d$ are mapped to $t + d$ by upward roundoff, as shown in Figure 4.9. Note that in this *quantized and granulized* representation, the outputs may be delayed by d in the worst case, but the change of state still propagates in zero time. For example, when a cell receives a flux change input from a neighbor cell

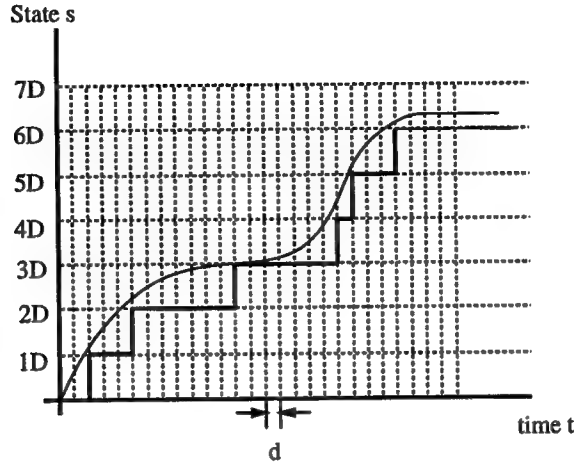


Figure 4.9: State space partitioning with granulized time axis.

at time t , this causes its state to change at the same time. Then the cell sends its updated states to all neighbor cells also at time t without delay, setting its *sigma* to 0 for this purpose.

4.4 Experimental Results

Figure 4.10 shows the elevation map of a watershed artificially created for experimentation. The target watershed is an array of 30×30 cells, each with dimension of $20m \times 20m$ ($400m^2$). We applied a 50 mm/hour rainfall to the whole watershed for ten hours and observed the behavior of the model during the period followed by a subsequent dry period. We compared the quantized DEVS models, with different state quanta and time granules to the simple discrete time DEVS models with various time steps. We chose the value 1.0 for a , b and P in Equation 4.2 and 4.3. The spatial evolution of flow was visually indistinguishable in all cases. However, to get

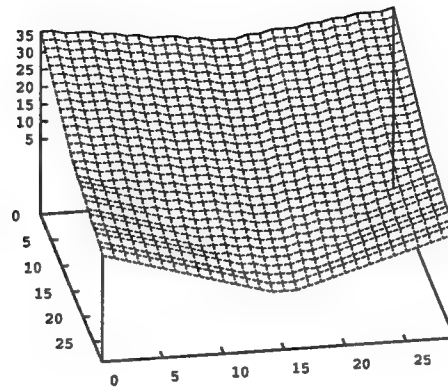


Figure 4.10: Elevation map of target watershed.

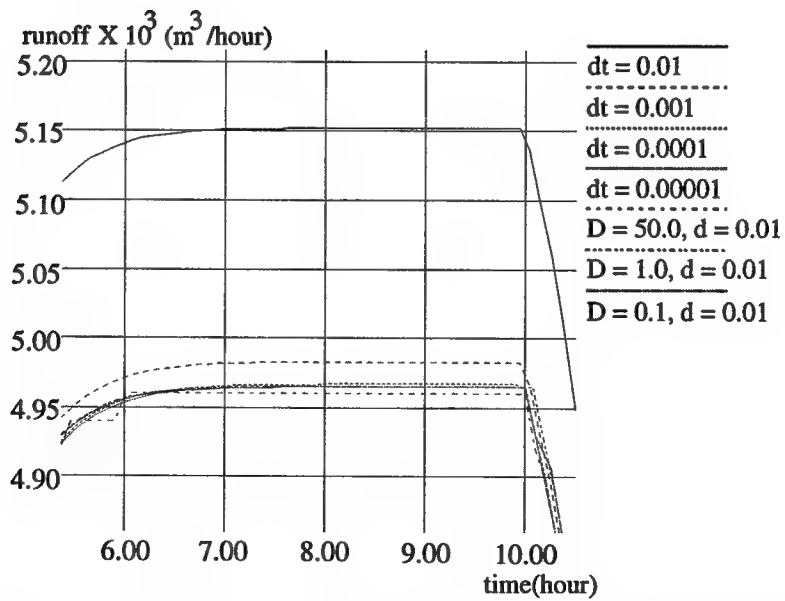


Figure 4.11: Simulation results: hydrographs in the steady state.

a quantitative estimate, we measured the runoff at the outlet (lowest point) over the 20 hour period. Figure 4.11 shows hydrographs obtained by these experiments near the steady state and Table 4.2 shows the runoff values of each model in the steady state.

Models	Runoff(m^3 /hour)	Difference from DM($dt = 0.00001$)
DM($dt=0.01$)	5,152.21	186.98
DM($dt=0.001$)	4,982.91	17.68
DM($dt=0.0001$)	4,966.86	1.63
DM($dt=0.00001$)	4,965.23	0.00
QM($D=50.0, d=0.01$)	4,960.00	-5.23
QM($D=1.0, d=0.01$)	4,966.00	0.77
QM($D=0.1, d=0.01$)	4,965.08	-0.15
QM($D=1.0, d=0.001$)	4,965.60	-0.77
QM($D=50.0, d=1.0e-8$)	4,960.00	-5.23

Table 4.2: Runoff of DEVS models in the Steady State. ($DM(dt)$ is the discrete time model, $QM(D, d)$ is the quantized DEVS model with quantum D and time granule, d , respectively.).

We assume that the discrete time DEVS model with the smallest time step (0.00001 hour) is the most accurate one. The third column of Table 4.2 is the steady state difference of each model from the 0.00001-step model baseline. The results show the quantized model with quantum 0.1 and time granule 0.01 is closer to the baseline than any other model.

Table 4.3 shows the execution times of the models on a Sparc-1000 processor. Simulation with the quantized model ($D=0.1$ and $d=0.01$) is 153 times faster than that of the discrete time model with time step 0.0001 (recall it is also more accurate) and 1,203 times faster than that of the discrete time model with time step 0.00001

Models	Iterations	Execution Time(sec)
DM(dt=0.01)	4,000	867
DM(dt=0.001)	40,000	8,353
DM(dt=0.0001)	400,000	92,235
DM(dt=0.00001)	4,000,000	726,557
QM(D=50, d=0.01)	35,100	375
QM(D=1.0, d=0.01)	24,200	523
QM(D=0.1, d=0.01)	29,100	604
QM(D=1.0,d=0.001)	87,300	2,345
QM(D=50.0,d=1.0e-8)	3,052,100	30,096

Table 4.3: Execution times of DEVS models.

(from which it differs very little). The results also show that the quantum size only affects the steady state error. The simulation time mainly depends on the time granule.

To analyze the source of this speedup we measured the number of events (sum of internal and external transitions) during simulation. Figure 4.12 and Figure 4.13 show the number of events sampled in 0.1 hour steps and the accumulated total number of events in log scale (base 10), respectively. The average number of events per iteration in the model QM(D=0.1,d=0.01) is approximately 6 % of that in DM(dt=0.00001).

As shown in Table 4.3 and Figure 4.13, we greatly reduce both the number of iterations and events by quantization and granulization without losing accuracy in the steady state. By this reduction we achieve about 1,000 fold speedup in simulation execution time.

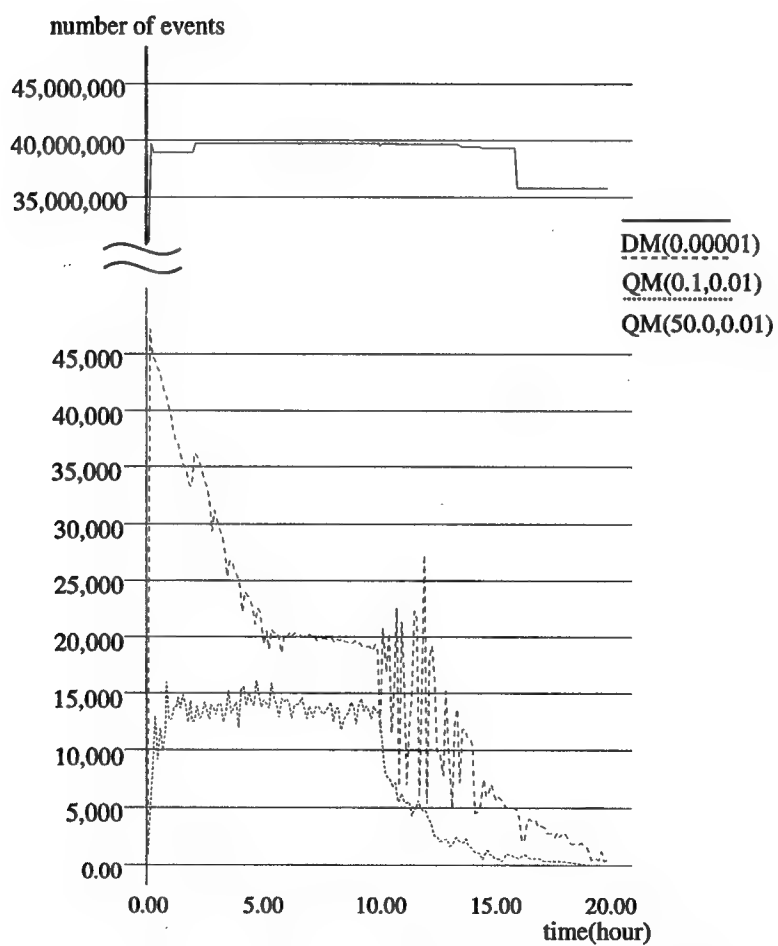


Figure 4.12: The Number of events for DM($dt=0.00001$), QM($D=0.1, d=0.01$) and QM($D=50.0, d=0.01$).

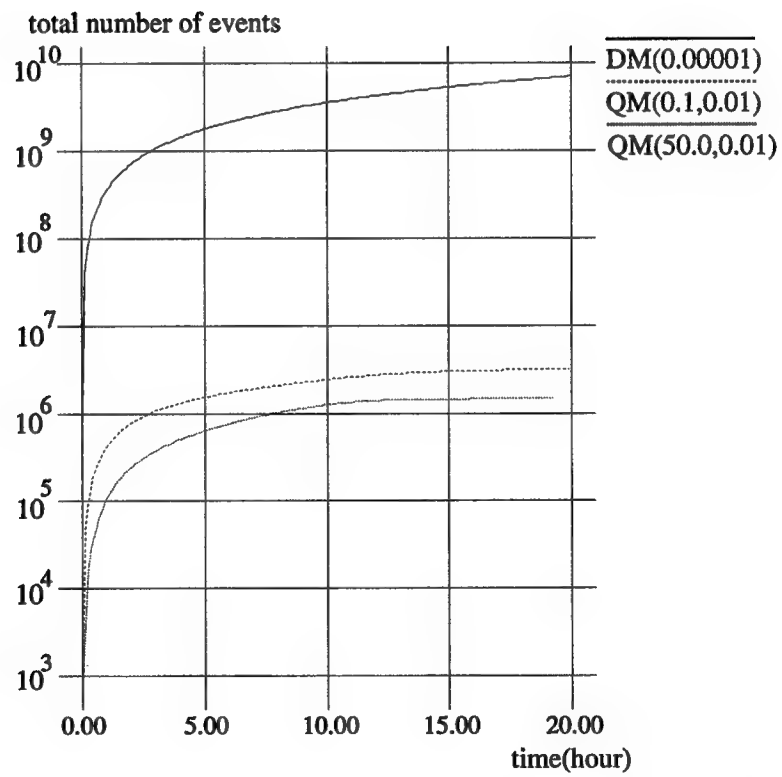


Figure 4.13: Accumulated total number of events in log scale (base 10).

CHAPTER 5

Optimization Example: Parameter Search for Watershed Models

The previous chapter showed that DEVS representation for spatially distributed systems could greatly reduce the simulation time. In the experiments, we arbitrarily chose the values for C , a , b and P in Equation 4.2 and 4.3. In this chapter, we'll show some examples of search for these parameters of watershed models in the proposed high performance simulation based optimization environment. We will also propose an approach to valid aggregation for spatially distributed systems to reduce the simulation time.

For experimental convenience, we employ KINEROS [48] as a target system, and show how to search the optimal values of the parameters in watershed models. The parameter search is considered only for a one-dimensional overland flow since KINEROS approximates overland flow as one-dimensional flow. The parameter P in Equation 4.3 can be considered as zero for overland flow since rainfall excess (water depth) difference between two cells is small enough compared to the elevation difference. By this assumption we have two advantages. First, we can reduce the communication overhead required to exchange rainfall excess data between the neighboring cells. Secondly, we can obtain an analytical form of spatial aggregation (will

be shown shortly). This assumption does not affect the speedups shown in Table 4.3 since communication overhead in both discrete time and DEVS models is reduced in the same ratio.

5.1 Search for the parameters a and b

In KINEROS, overland flow is approximated as one-dimensional flow process in which the runoff is given by:

$$Q = \alpha h^m. \quad (5.1)$$

Four options for α and m are provided in KINEROS. In the experiments, we chose the Manning hydraulic resistance law. In this option,

$$\alpha = \frac{1.49S^{0.5}}{n}, \quad (5.2)$$

and $m = \frac{5}{3}$, where S is the slope and n is the Manning roughness coefficient. The range of Manning's roughness coefficient is from 0.01 to 0.68 which correspond to concrete and grassland, respectively [48].

We use the following steps to obtain the optimal values for the parameters a and b of the watershed model.

- Step 1: Obtain the behavior of KINEROS for three simple planes with three different slopes for $n = 0.1$.

- Step 2: Search for the best values of the parameters, C , a and b of watershed models with three different resolutions compared to the target behavior obtained in Step 1.

In Step 1, we obtain the target behavior (hydrographs) for three 500 m long planes (the slopes of the planes are 0.01, 0.05 and 0.1, respectively) using KINEROS. Simulations covered 3 hours of real time for the 2 hour long, 30 mm/hour rainfall event. In Step 2, we run optimization to find the optimal values for C , a and b for three watershed models with different resolutions (10, 20 and 50 cells). The criterion for optimization is to minimize the maximum absolute runoff error. The search ranges are 1.0 – 10000.0 for C , and 0.0 – 3.0 for a and b . Each parameter is decoded as a 16-bit binary string in the GA optimizer.

model resolution	C	a	b	maximum runoff error (mm/hour)	optimization time (sec)
10 cells	336.9	0.52	1.74	1.3	485
20 cells	285.1	0.50	1.74	0.73	1,225
50 cells	376.7	0.48	1.62	0.48	4,193

Table 5.1: Results of parameter search for a and b (optimization time is measured for 20,000 GA iterations on the CM-5 with 256 nodes).

As shown in Table 5.1, the GA optimizer found the values, 0.48 for a , 1.62 for b and 376.7 for C . The watershed model has a resolution of 50 cells. Its maximum runoff error was 0.48 (mm/hour) which was 1.6 % of the steady state runoff (=30.0 mm/hour). This optimization takes 4,193 seconds for the 50 cell DEVS model on the

CM-5 with 256 nodes, while the estimated execution time on a Sparc-1000 workstation is about 6 days. This indicates that even a simple parameter search based on simulation is impractical without help of the high performance optimization environment proposed in the report.

Figure 5.1 shows the hydrographs of the DEVS model with 50 cells compared to those of KINEROS for three different slopes. Figure 5.2 also shows the hydrographs of the DEVS model with 50 cells compared to those of KINEROS for a randomly generated rainfall event. In the latter case, the maximum runoff error was 0.53. This result shows that the values obtained for one simple rainfall pattern can be used for other rainfall patterns.

5.2 Search for the C range

In the previous section, we obtained the optimal values for parameters a and b of the watershed model. However, search for the optimal value of C for each cell in the watershed model requires the range of the parameter C . Based on the Manning's coefficient table given in the literature [48], it's only necessary to find the value of C for the concrete and grassland which correspond to Manning's coefficients, 0.01 and 0.68, respectively. With $a = 0.48$ and $b = 1.62$ obtained in the previous section, we search for the optimal values of C for the concrete and grassland with the resolution of 50 cells.

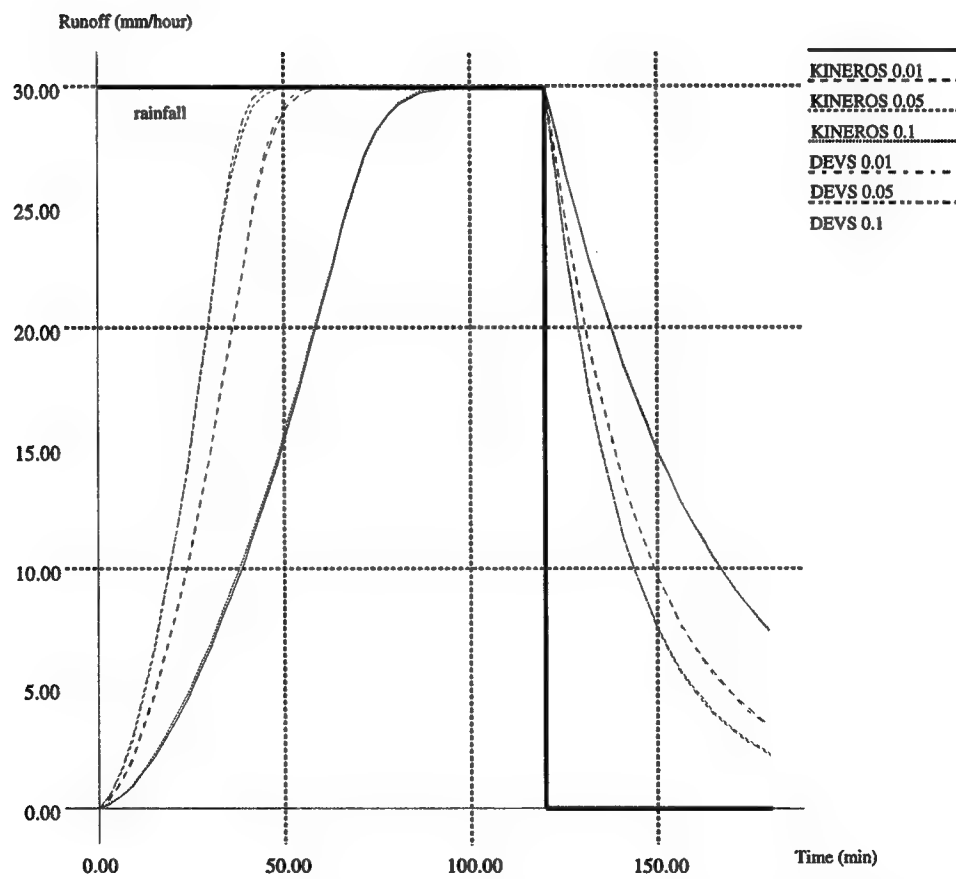


Figure 5.1: Hydrographs of the optimized watershed model and KINEROS for the planes with the slopes of 0.01, 0.05 and 0.1.

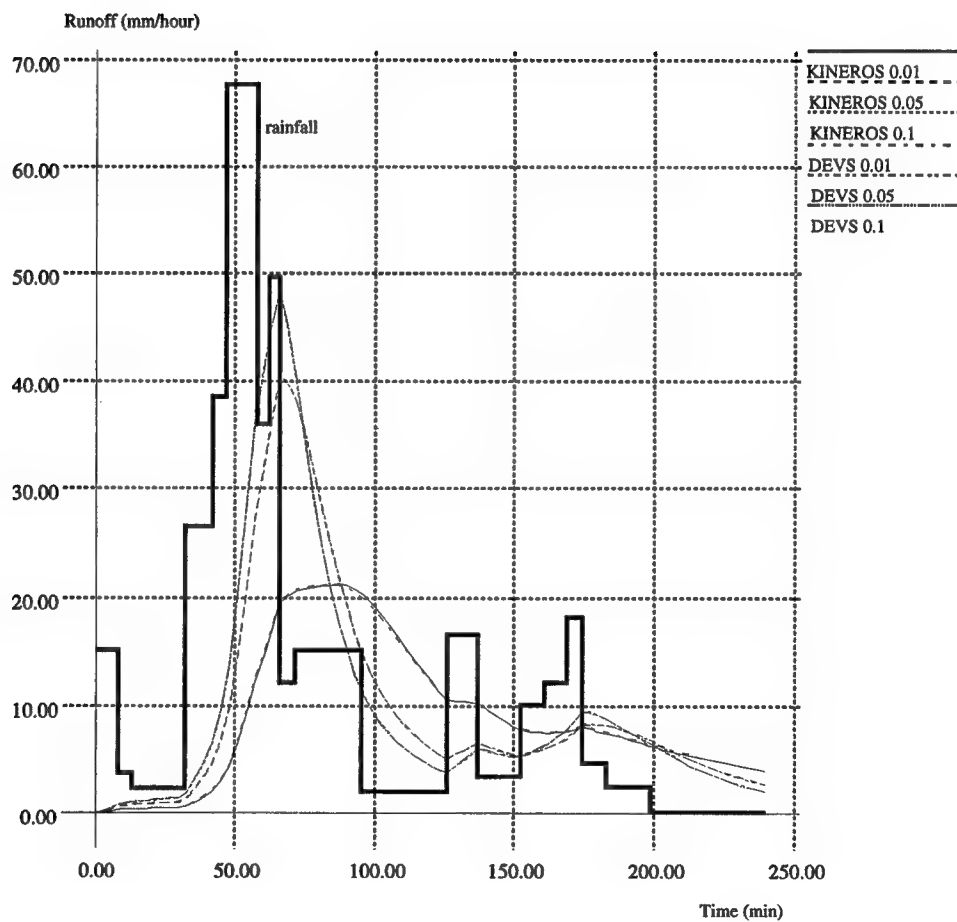


Figure 5.2: Hydrographs of the optimized watershed model and KINEROS for a randomly generated rainfall event.

The C range obtained in this experiment was from 65.6 (for grass) to 3213.1 (for concrete) with the maximum runoff error of 0.57 (mm/hour). The GA optimizer found this range within 1,000 iterations. Figure 5.3 and 5.4 show the hydrographs generated by the DEVS model of the grassland with the optimized parameters for different rainfall patterns compared to KINEROS.

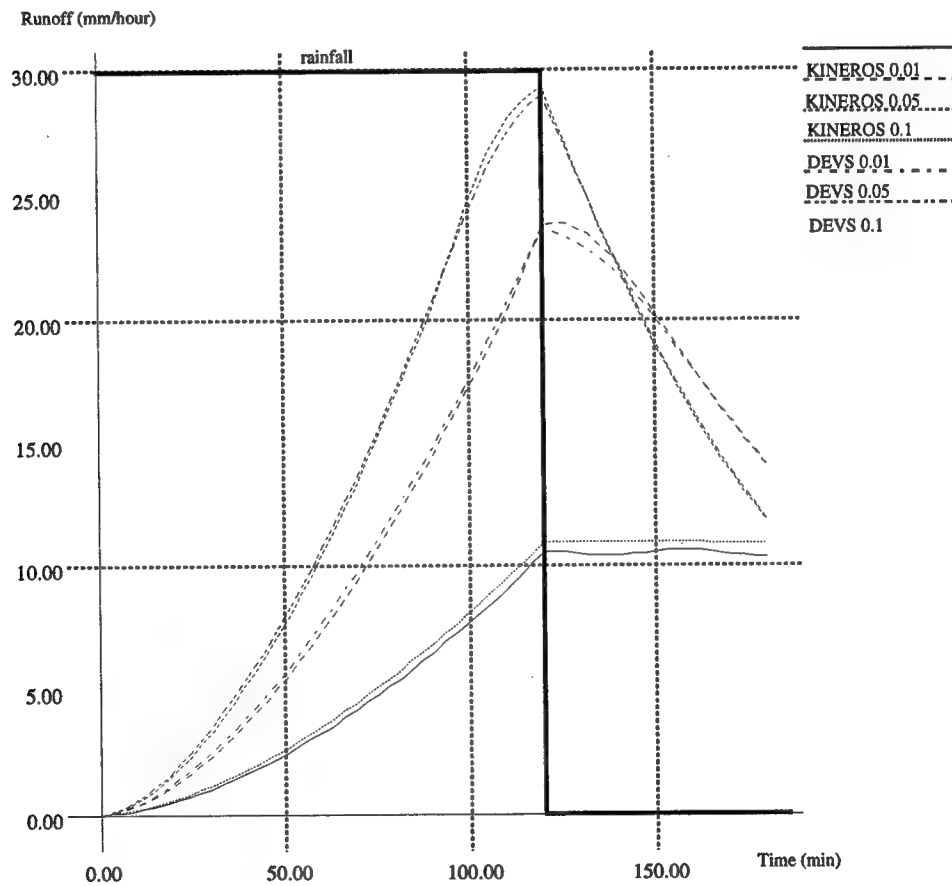


Figure 5.3: Hydrographs of the optimized watershed model and KINEROS for slope 0.01, 0.05 and 0.1.

Another experiment was conducted to show how the resolution of watershed affects the modelling accuracy, the optimal value of C and the search time. The experiment

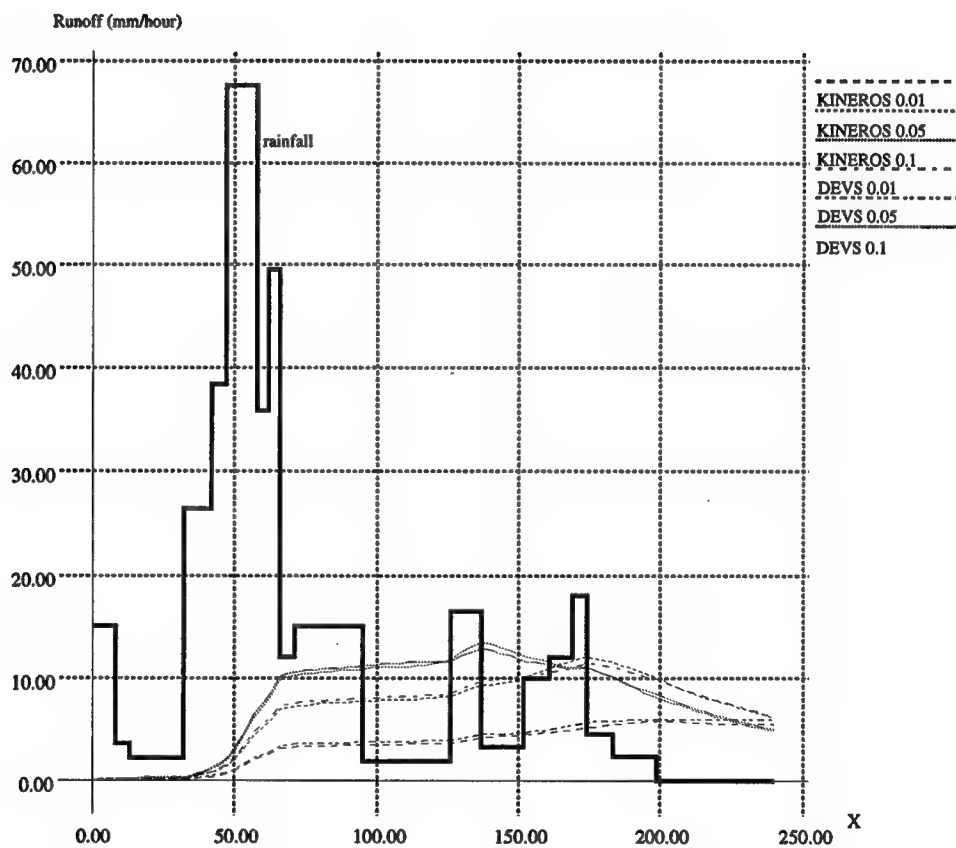


Figure 5.4: Hydrographs of the optimized watershed model and KINEROS for randomly generated rainfall event.

found the optimal value C of the watershed model with seven different resolutions (1, 2, 4, 8, 16, 32, 64 cells) for the target watershed in the previous section ($n = 0.1$). As shown in Table 5.2, the optimal value of C depends on the resolution of the watershed model. However, the optimal resolution that minimizes the runoff error is not the highest one. This result seems to fly in the face of the usual assumption that higher resolution is better. It may be that there is an optimal level of resolution for a given behavior. We have not probed deeper to resolve why this should be so. (The situation is reminiscent of numerical integration methods where decreasing step size may sometimes increase error due to propagation of roundoff error introduced at each step (over more steps as resolution increases)).

model resolution	C	maximum runoff error (mm/hour)	optimization time (sec)
1 cell	659.0	4.30	54
2 cells	538.3	3.33	65
4 cells	466.9	2.52	136
8 cells	419.7	1.71	361
16 cells	396.7	1.03	986
32 cells	381.3	0.44	2,412
64 cells	369.5	0.52	5,655

Table 5.2: Results of parameter search for C (although the best values are obtained within 1,200 GA iterations for all cases, the optimization time is measured for 20,000 iterations on the CM-5 with 256 nodes).

5.3 Search for C surface roughness

Two previous sections obtained the optimal values for a , b and the range of C by the experiments. This section addresses the problem of how to search for the

best value of C within each cell of the watershed model for a given target watershed. Although the GIS data base can provide roughness information to some extent, the parameter search or tuning is still required to obtain an optimal C value for each cell since it depends on the resolution of watershed models as shown in Table 5.2.

However, parameter search for large watersheds takes a tremendous amount of time as shown in Table 5.3. For example, one simulation run of the DEVS model with a quarter million cells takes about three days on a Sparc-2 processor. Thus parameter search requiring 20,000 GA iterations takes about 1.5 centuries. Moreover, this is calculated under the assumption that GAs can optimize a quarter million parameters within 20,000 iterations.

Number of cells	Execution time/run (sec)	Execution time/20,000 runs
961 cells (31×31)	242	2 months
3,696 cells (63×63)	1,486	1 year
16,129 cells (127×127)	8,084	5 years
65,025 cells (255×255)	49,319	3 decades
261,121 cells (511×511)	231,177	1.5 centuries

Table 5.3: Execution times of watershed models on a Sparc-2 processor for a 2 hour rainfall event.

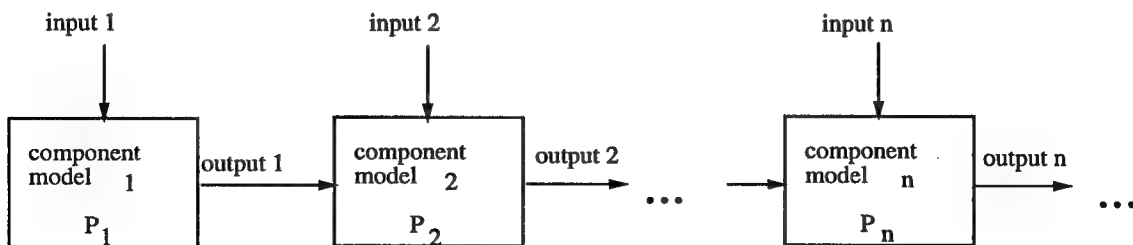


Figure 5.5: A system that has some number of component models connected in cascade. (P_n is a parameter in component model n .)

To solve this problem, consider the system with n component models connected in cascade shown in Figure 5.5. Assume that each component model n has one parameter to be optimized. It is clear that parameter optimization for the component model 1 is not dependent on any other component models when the output of model 1 does not depend on the states of any other model. With the optimized component model 1, parameter optimization for the component model 2 becomes independent of other component models in the same condition. Thus we can transform a big search problem into many smaller search problems. This method, *incremental optimization*, can be applied to watershed models when the direction of water flow does not change during a rainfall event. This incremental optimization may not find the global optimal solutions, but it can greatly reduce the optimization time. The estimated optimization time for the 511×511 watershed model with this method is about 2 days on the CM-5 with 512 nodes. That is, we can reduce 2 years of optimization time to 2 days. Estimation is based on the assumption that it takes 1,000 iterations to tune one parameter of one cell (the execution time of 1,000 GA iterations for the watershed model with 1 cell is measured as about 0.7 seconds).

To generalize the incremental optimization, consider the system in Figure 5.6. Assume that the system has $n \times n$ blocks of models and each block has $m \times m$ models (that is, the system has $N \times N$ models in total for $N = n \times m$). If the input/output behavior of each model inside the blocks is available, an $N \times N$ parameters search problem can be reduced to $N \times N$ single parameter search problems as explained

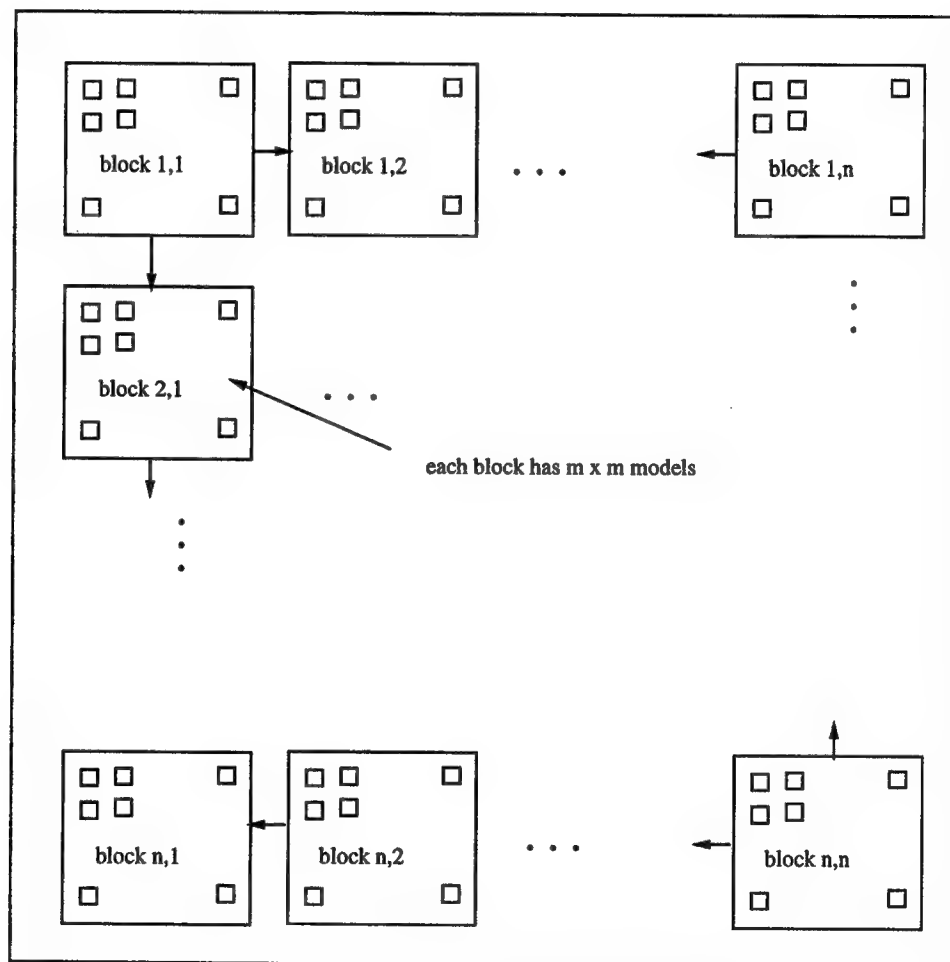


Figure 5.6: A system that has $n \times n$ blocks of models and each block has $m \times m$ models.

above. However, if only the input/output behavior for each block is available, the $N \times N$ parameter search problem is transformed into $n \times n$ ($m \times m$)-parameter search problems. The reduction of optimization time for this case depends on the complexity of GAs. Let the complexity of GAs be $O(l^k)$, where l is the chromosome length and k is a constant. Then the complexity of the $N \times N$ parameter search problem is $O((N \times N)^k)$ and that of the $n \times n$ ($m \times m$)-parameter search problems is $O(n \times n \times (m \times m)^k)$. From $N = n \times m$, we can achieve $(n \times n)^{k-1}$ fold speedups. The constant k depends on the problem to be optimized by GAs.

An experiment is conducted to show the effectiveness of incremental optimization and to get the complexity of GA optimization for watershed models. We randomly create the target watersheds (one-dimensional DEVS models) with 4 different number of cells. The slopes and C values are chosen between 0.01 and 0.1, and between 50.0 and 3500.0, respectively. We run GA optimization until it finds the watershed model with the maximum runoff error of 1.5 mm/hour (which is 5 % of the steady state runoff). Table 5.4 and 5.5 show the results of the experiment. As shown in the tables, the incremental optimization markedly reduced execution time. Table 5.4 also shows the complexity of the GA optimizer for watershed model. The constant k in $O(l^k)$ is definitely more than 2.

number of cells	GA iterations	maximum runoff error	Execution time(sec)
4	508	1.00	40
8	11,589	1.00	87
16	60,200	1.50	780
32	312,239	1.48	7,650

Table 5.4: Results of global optimization (execution times are measured on the CM-5 with 256 nodes).

number of cells	GA iterations	maximum runoff error	Execution time(sec)
4	352	1.00	5
8	630	1.00	7
16	920	1.50	9
32	1,215	1.48	11

Table 5.5: Results of global optimization (execution times are measured on a Sparc-1000).

5.4 Spatial Aggregation

The previous chapter showed that DEVS abstraction could greatly reduce the simulation time. However, simulations of large watershed models still require a powerful computer such as the CM-5 or IBM SP2. Table 5.3 shows that one simulation run for 511×511 cells on one Sparc-2 processor takes about 3 days. The table also indicates that lowering the spatial resolution with some allowable error can greatly reduce simulation time. For example, lowering the spatial resolution by a factor of 4, from 511×511 to 127×127 , reduces simulation time by the factor of 30 as shown in Table 5.3. To do so, we develop an approach to valid aggregation of spatial watershed models based on *parameter morphism*. Aggregation of the states for variable-resolution modeling was studied by Davis [49]. Abstraction of DEVS models by aggregation

was studied by Zeigler [7]. As far as we know, however, there has been no research related to aggregation of spatially distributed systems.

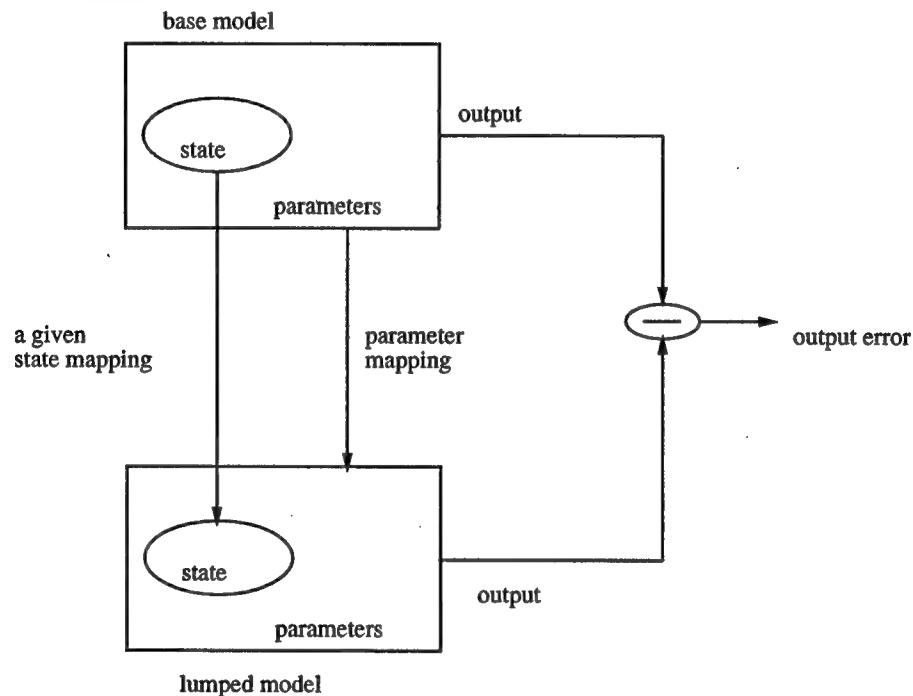


Figure 5.7: Parameter morphism.

Parameter morphism is defined as a parameter mapping that yields a lumped model, a valid abstraction, of a base model in some experimental frame. As shown in Figure 5.7, parameter morphism preserves the output behaviors of the base and lumped model with some allowable error. There are two methods to derive parameter morphism — a GA optimization based method and an analytical method. In the former, parameter morphism can be obtained using GA optimization as shown in Figure 5.8. In the latter, parameter morphism can be obtained analytically (an

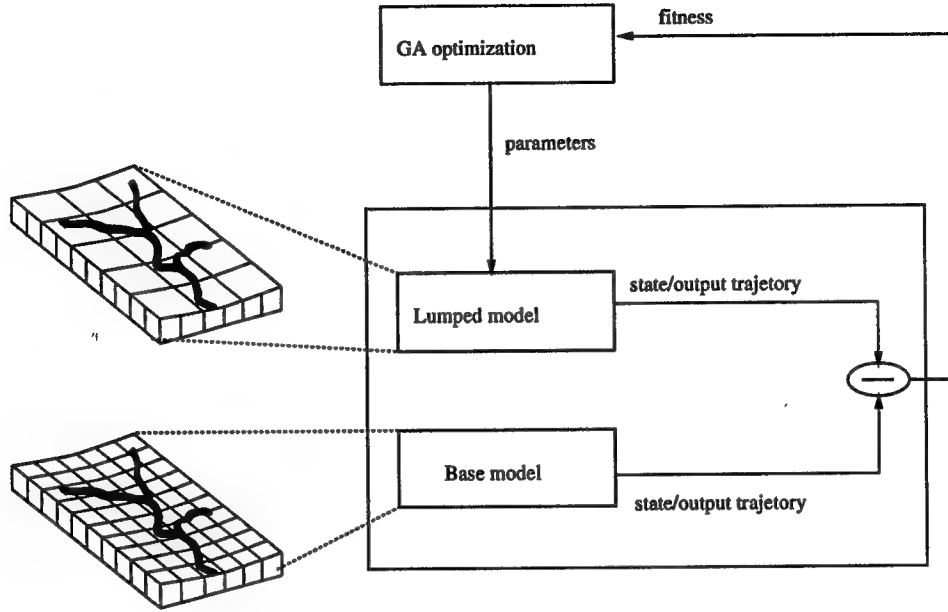


Figure 5.8: Parameter morphism by GA optimization.

example will be shown shortly). We will not show the example of the former method since it's a trivial GA optimization problem.

Consider a simple one-dimensional watershed shown in Figure 5.9. We will aggregate pairs of adjacent cells, i and $i + 1$, into single cells. The problem is how to obtain a surface roughness value C_j of the low resolution model from the values, c_i and c_{i+1} of a high resolution model. With the assumption of $P = 0.0$, one possible set of reasonable constraints for this spatial aggregation is:

$$\begin{aligned}
 Q_j &= \frac{q_{i+1}}{2}, \\
 R_{x_j} &= \frac{r_{x_i} + r_{x_{i+1}}}{2}, \\
 S_j &= \frac{s_i + s_{i+1}}{2}.
 \end{aligned} \tag{5.3}$$

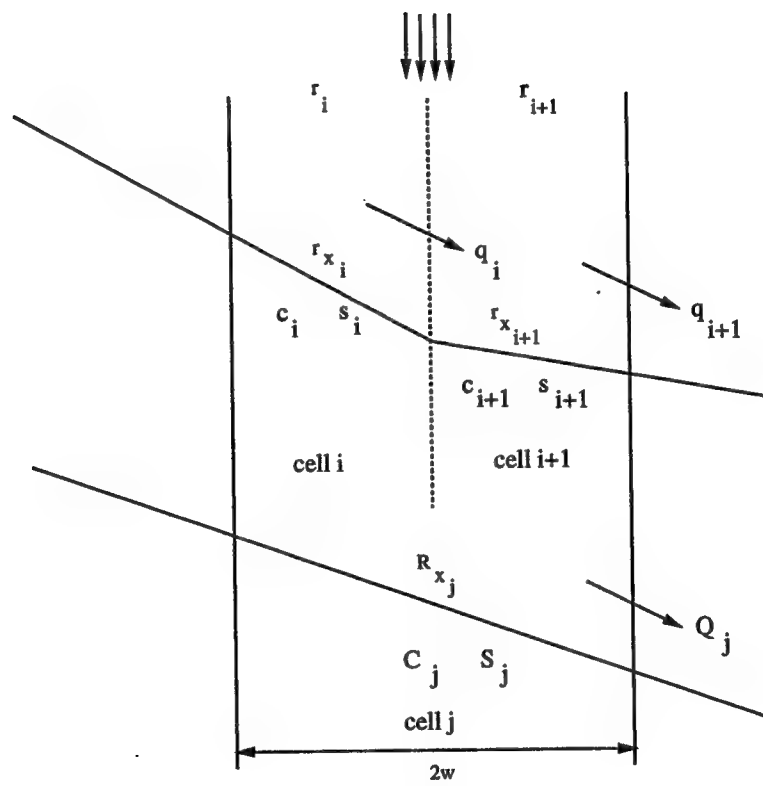


Figure 5.9: Spatial aggregation for one-dimensional flow.

From equations 4.2 and 5.3, the surface roughness C_j of the low resolution model is calculated by:

$$C_j = c_{i+1} \left(\frac{2s_{i+1}}{s_i + s_{i+1}} \right)^a \left(\frac{2r_{x_{i+1}}}{r_{x_i} + r_{x_{i+1}}} \right)^b. \quad (5.4)$$

As shown in Equation 5.4, the parameter C_j depends on rainfall excess of the high resolution model. In other words, C_j must be time varying to satisfy the constraints given by Equation 5.3. However, in the steady state,

$$q_{i+1} = q_i + r_{i+1}, \quad (5.5)$$

where r_{i+1} is the rainfall intensity for cell $i + 1$. Using equations 4.2 and 5.5, in the steady state we have:

$$\frac{c_{i+1}s_{i+1}^a r_{x_{i+1}}^b}{W} = \frac{c_i s_i^a r_{x_i}^b}{W} + r_{i+1}, \quad (5.6)$$

where W is the width of a cell in the high resolution model. Under zero rainfall, we have $r_{i+1} = 0$, and

$$r_{x_i} = \left(\frac{c_{i+1}s_{i+1}^a}{c_i s_i^a} \right)^{\frac{1}{b}} r_{x_{i+1}}. \quad (5.7)$$

From equations 5.4 and 5.7, C is calculated as:

$$C_j = c_{i+1} \left(\frac{2s_{i+1}}{s_i + s_{i+1}} \right)^a \left(\frac{2}{1 + \left(\frac{c_{i+1}s_{i+1}^a}{c_i s_i^a} \right)^{\frac{1}{b}}} \right)^b. \quad (5.8)$$

Although steady state and zero rainfall conditions were employed to derive Equation 5.8, we can experiment to see how well this parameter mapping works in general.

5.4.1 Experimental Results

In this section, we will present some experimental results for spatial aggregation based on the proposed parameter mapping.

To show how much we can reduce spatial resolution, we did the following experiments.

- Step 1: Create the base (high resolution) model with a randomly generated C value (within the range found in the previous section) and slope for each cell.
- Step 2: Obtain the lumped models using the parameter mapping given by Equation 5.8 and measure the error between the lumped and base models.

In Step 1, we create the base model with 128 cells for a 2,560 m long watershed and randomly assign the slope s and C values to each cell ($0.01 \leq s \leq 0.1$, $60.0 \leq C \leq 3500.0$). In Step 2, we apply the parameter mapping to the base model and get a lumped model. We repeat the lumping process until we get a lumped model with only 1 cell. We measure the runoff error between the base and lumped models in each lumping step. The maximum absolute error is the criterion for comparison.

The tables 5.6, 5.7 and 5.8 show the results for three different randomly generated base models. Simulations covered 3 hours of real time with 2 hour long 30 mm/hour rainfall event. The results show that we can reduce the spatial resolution by a factor of 8 with less than 10 % error from the steady state value of runoff. Figure 5.10 shows the hydrograph (runoff) of the base and lumped models at the outlet of the

Models	maximum runoff error (mm/hour)
base (128 cells)	0.00
lumped (64 cells)	0.50
lumped (32 cells)	1.42
lumped (16 cells)	2.72
lumped (8 cells)	4.28
lumped (4 cells)	6.28
lumped (2 cells)	8.78
lumped (1 cells)	10.44

Table 5.6: Runoff error between base and lumped model for base model 1 (simulated for 3 hours with a 2 hour long 30 mm/hour rainfall event).

Models	maximum runoff error (mm/hour)
base (128 cells)	0.00
lumped (64 cells)	0.60
lumped (32 cells)	1.37
lumped (16 cells)	2.37
lumped (8 cells)	3.75
lumped (4 cells)	5.62
lumped (2 cells)	8.12
lumped (1 cells)	10.19

Table 5.7: Runoff error between base and lumped model for base model 2 (simulated for 3 hours with a 2 hour long 30 mm/hour rainfall event).

Models	maximum runoff error (mm/hour)
base (128 cells)	0.00
lumped (64 cells)	0.51
lumped (32 cells)	1.23
lumped (16 cells)	2.26
lumped (8 cells)	3.70
lumped (4 cells)	5.42
lumped (2 cells)	7.42
lumped (1 cells)	9.84

Table 5.8: Runoff error between base and lumped model for base model 3 (simulated for 3 hours with a 2 hour long 30 mm/hour rainfall event).

watershed. Smaller error after the rain stops is consistent with the assumptions behind parameter mapping.

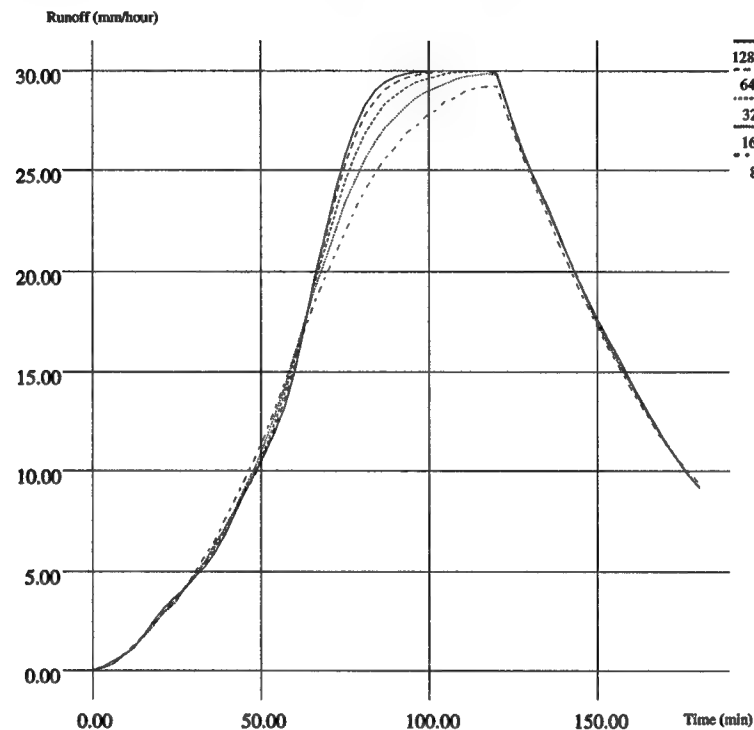


Figure 5.10: Runoff of the base model and lumped models

Models	Execution time (sec)
base (128 cells)	60.1
lumped (64 cells)	26.3
lumped (32 cells)	10.2
lumped (16 cells)	4.0

Table 5.9: Execution times of the base and lumped models on a Sparc-2 processor.

In Table 5.9, execution time of each model measured on a Sparc-2 processor shows we can achieve about 15 fold speedup by lumping with less than 10 % error.

5.4.2 Multiresolution search based on parameter morphism

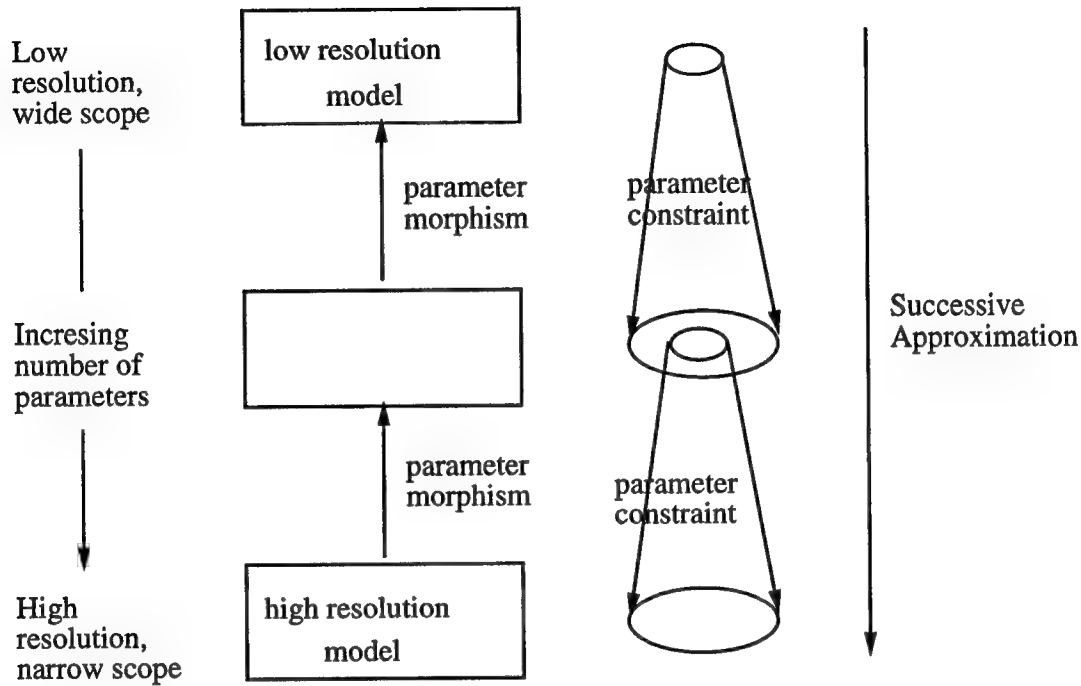


Figure 5.11: Multiresolution search strategy.

As shown in the previous sections, parameter morphism can greatly reduce the simulation time of watershed models. This section addresses the optimization time reduction for watershed models using a multiresolution search. As depicted in Figure 5.11, the multiresolution search strategy can obtain the optimal values of the parameters in the high resolution model if the parameter values of the high resolution models are located in the vicinity of those of low resolution models. That is, we first search for the optimal parameters in the low resolution model. Then we search for the parameters in the high resolution model in the vicinity of the values. This process is repeated until the model with the desired level of resolution is found. This method

may reduce optimization time. But we leave the research related to this topic as future work.

CHAPTER 6

Conclusions

This report proposed a high performance simulation based optimization environment capable of supporting the design and modeling of large scale systems with natural and artificial components at high levels of resolution.

We demonstrated the advantage of using the DEVS formalism to represent large scale continuous system models in efficient high fidelity. For example, in the case of watershed behavior discussed in the report, traditional approaches based on partial differential equations decompose the watershed into "parking lots", each with builtin channel flow – without such coarse representation, such simulations would take months or years to complete. In contrast, our high resolution model allows channel flows to "emerge" from the underlying water dynamics and landscape topography. Figure 6.1 is an elevation map of a real watershed, Brown's Pond. Figure 6.2 shows the distribution of runoff at some time after a uniform rainfall. One can see that channels have formed that are clearly correlated with the topography.

It should be noted that hybrid models containing both discrete and continuous components offer an attractive alternative and are undergoing intensive research [50]. However, without mapping the continuous parts to DEVS they can not exploit the

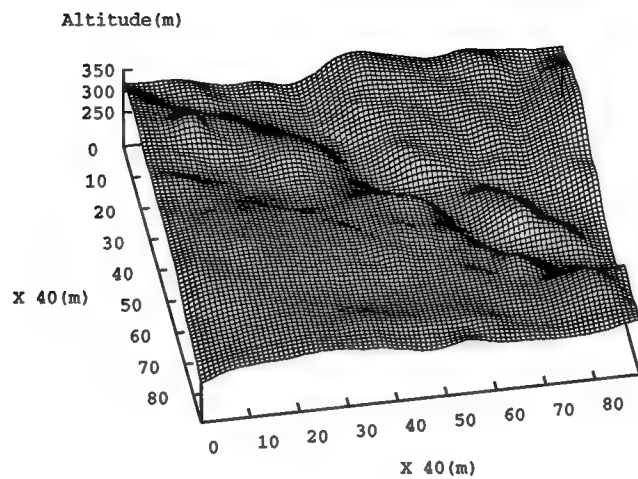


Figure 6.1: Brown's pond elevation map.

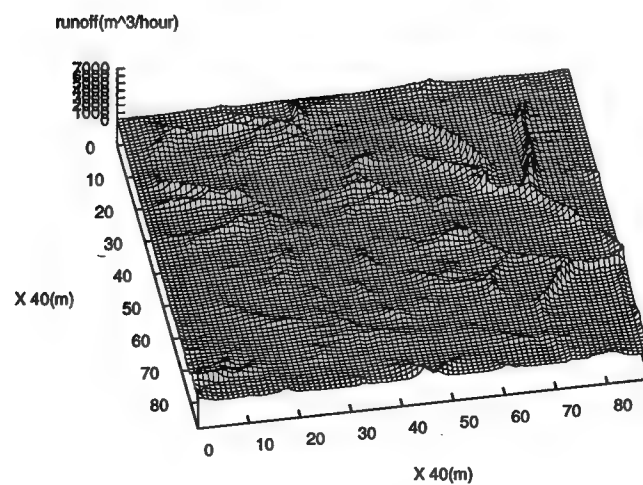


Figure 6.2: Brown's pond runoff (m^3/hour) after 2 simulated hours (1 hour after end of 1 hour long rainfall).

thousand-fold speedups necessary to achieve feasible optimizations of complex systems. As shown in the report, the quantized/granulized DEVS model achieved the thousand-fold speedups compared to the discrete time model.

The searcher layer has employed Genetic Algorithms to provide generic robust search capability. We have developed a class of parallel Genetic Algorithms, called Distributed Asynchronous Genetic Algorithm (DAGA), which provides the speed required for simulation based optimization of large scale systems with various applications.

Coded in the object-oriented language, C++, the proposed environment runs on both serial and parallel computing platforms [51]. The universality of the the DEVS modelling formalism, the portability of the C++ implementations, and the robustness of the GA searcher layer are intended to facilitate widespread use of the environment.

We have presented actual experiments that show how each of the sources — DEVS representation, and distributed GA-based search can individually achieve thousand fold speedups. With the DEVS representation, and the combined simulation/searcher layers each affording a thousand fold speedup, taken together, these sources of high performance can achieve at least a million fold speedup over current workstation performance levels. As indicated, performance increases of such scale will make possible some ambitious studies that are not feasible today.

This report also proposed an approach to valid aggregation of spatially distributed systems based on parameter morphisms, and showed its validity and reduction of

simulation time for watershed models. However, several topics such as spatial aggregation for two-dimensional watershed models, parameter search for large watershed models using GA optimization based on multiresolution search strategy, and validation of watershed models against real watersheds, still remain as future work.

Appendix A

Fuzzy Systems

The basic idea of fuzzy systems centers around the *labeling* process, in which the reading of a sensor is translated into a label as done by human expert controllers [52]. With expert supplied membership functions for labels, a reading of a sensor can be *fuzzified* and *defuzzified*. It is important to note that the transitions between labels are not abrupt and a given reading might belong to several label regions.

However, the fuzzification and defuzzification processing need not be sequential. The input signal can be fuzzified/defuzzified simultaneously by matching membership functions. Therefore fuzzy control processing can be adapted to a parallel neural network structure where each *neuron* represents functions (fuzzy membership) and each *link* represents the weight of a fuzzy rule.

Figure A.1(a) shows the structure of the fuzzy system and its fuzzy subspace (Figure A.1(b)) [38]. In this example, 5 fuzzy regions are defined for the inputs and output.

While an earlier fuzzy systems [53, 54] was implemented in rule-based form (*if-then*), the fuzzy system employs parallel inferencing network structure. Due to the parallel fuzzification/defuzzification scheme, the fuzzy system can improve real-time

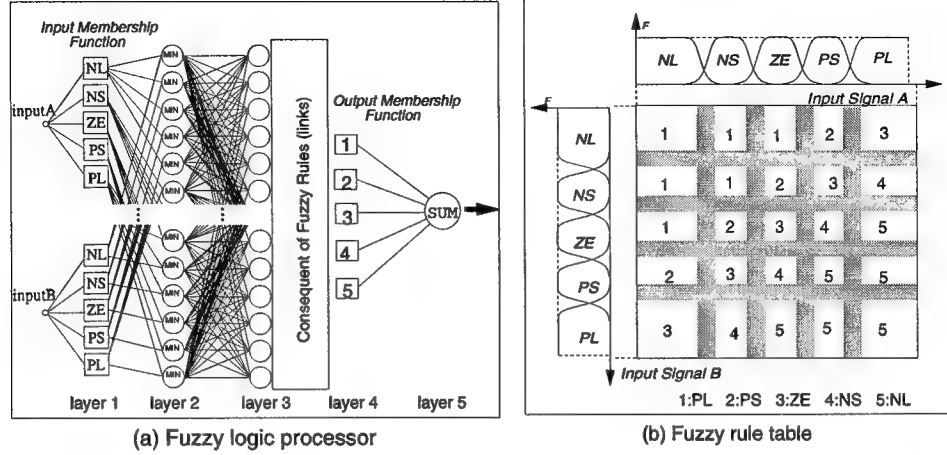


Figure A.1: Fuzzy inference network and fuzzy subspaces

performance of the control system for practical applications. The operations of layers in fuzzy inference network are,

Layer 1 Every node i in the first layer has a node function, $O_i^1 = \mu_{A_i}(x)$. O_i^1 is the membership function of A_i (a linguistic label such as positive small, negative large, etc.) and it specifies the degree to which x given satisfies the quantifier A_i .

Layer 2 A node in the second layer performs the generalized AND operation. w_i represents the firing strength of rule- i , $w_i = \min(\mu_{A_i}(x), \mu_{B_i}(x))$.

Layer 3 Every node computes the ratio of the i -th rule's firing strength to the sum of the firing strengths of all the rules. $\bar{w}_i = \frac{w_i}{\sum_{k=0}^n w_k}$.

Layer 4 Every node computes the defuzzified value of each rule i . $O_i^4 = \bar{w}_i f_i$, where f_i is the defuzzified value for rule i .

Layer 5 This node computes the overall output as the summation of all incoming signals. $O^5 = \sum_{i=0}^n O_i^4 = \sum_{i=0}^n \bar{w}_i f_i$.

In order to find well-performing fuzzy membership functions without help of human expertise, it is necessary to employ computer-aided optimization. Since tuning the membership functions requires adjusting many parameters simultaneously, hill-climbing search methods would suffer from the complexity of the search space.

For this reason, GAs were employed to find optimal membership functions and rules in this research.

REFERENCES

- [1] Committee on Future Technologies, "Commercial Multimedia Technologies and the 21st Century Army: A Technology Management Strategy," tech. rep., National Research Council, Washington DC, 1996.
- [2] PVM project team, "PVM User Survey Results." posted on comp.parallel.pvm, 1994.
- [3] J. Kim and B. P. Zeigler, "Hierarchical Distributed Genetic Algorithms in an Intelligent Machine Architecture," *IEEE Expert Magazine*, vol. 11, pp. 76-84, June 1996.
- [4] J. Kim, Y. Moon, and B. P. Zeigler, "Designing Fuzzy Neural Net Controllers using Genetic Algorithms," *IEEE Control Magazine*, vol. 15, no. 3, pp. 66-72, 1995.
- [5] A. Louri, H. Sung, Y. Moon, and B. P. Zeigler, "An Efficient Signal Distinction Scheme for Large-scale Free-space Optical Networks Using Genetic Algorithms," in *Photonics in Switching: Topical Meeting, OSA*, Salt Lake City, Utah, pp. 90-92, Mar. 12-17 1995.
- [6] B. P. Zeigler, Y. Moon, V. L. Lopes, and J. Kim, "DEVS Approximation of Infiltration Using Genetic Algorithm Optimization of a Fuzzy System," *Journal of Mathematical and Computer Modeling*, vol. 23, pp. 215-228, June 1996.
- [7] B. P. Zeigler, *Theory of Modelling and Simulation*. New York: John Wiley, 1976.
- [8] Y. C. Ho, "Special issue on discrete event dynamic systems," *Proceedings of the IEEE*, vol. 77, no. 1, 1989.
- [9] B. P. Zeigler, *Multifaceted Modelling and Discrete Event Simulation*. London: Academic Press, 1984.
- [10] B. P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. San Diego, CA: Academic Press, 1990.

-
- [11] B. P. Zeigler and W. H. Sanders, "Preface to Special Issue on Environments for Discrete Event Dynamic Systems," *Discrete Event Dynamic Systems: Theory and Application*, vol. 3, no. 2, pp. 110–119, 1993.
- [12] B. P. Zeigler, T. G. Kim, H. Praehofer, and H. S. Song, "DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems," in *Lecture Notes in CS* (P. Antsaklis and A. Nerode, eds.), pp. 529–551, Springer-Verlag, 1996.
- [13] H. Praehofer and B. P. Zeigler, "On the Expressibility of Discrete Event Specified Systems," in *Proceedings of CAST (Computer Aided Systems Theory) Conference, Ottawa, Canada, May 1994*, Springer Verlag, 1996. (will appear).
- [14] H. Praehofer, "System Theoretic Foundations for Combined Discrete-Continuous System Simulation," Ph.D. dissertation, Johannes Kepler University of Linz, Linz, Austria, 1991.
- [15] A. Chow and B. P. Zeigler, "Revised DEVS: a Parallel, Hierarchical, Modular Modeling Formalism," in *Winter Simulation Conf.*, 1994.
- [16] Y. K. Cho, "Parallel Implementation of Container using Parallel Virtual Machine," Ph.D. dissertation, The University of Arizona, Tucson, Arizona, 1995.
- [17] V. L. Sunderam and et. al., "The PVM Concurrent Computing System: Evolution, Experience, and Trends," *Parallel Computing*, vol. 20, no. 4, pp. 531–545, 1994.
- [18] B. P. Zeigler, Y. Moon, D. Kim, and G. Ball, "High Performance Modelling and Simulation: Progress and Challenges," *IEEE Computational Science and Engineering*, 1996. accepted.
- [19] J. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [20] D. Whitley and T. Starkweather, "GENITOR II : A Distributed Genetic Algorithm," *Journal of Expert Theory and Artificial Intelligence*, vol. 2, pp. 189–214, Jan. 1994.

- [21] V. S. Gordon and D. Whitley, "Serial and Parallel Genetic Algorithms as Function Optimization," in *Proceeding of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, pp. 177-190, Univ. of Illinois Urbana-Champaign, July 1993.
- [22] Z. Michalewicz, *Genetic Algorithm + Data Structure = Evolution Programming*. New York: Springer-Verlag, 1992.
- [23] T. Back and H. P. Schwefel, *An Overview of Evolutionary Algorithms for Parameter Optimization*, vol. 1, pp. 1-23. MIT Press, 1993.
- [24] M. Srinivas and L. M. Patnaik, "Genetic Algorithms: A Survey," *Computer*, vol. 27, pp. 17-26, June 1994.
- [25] J. L. R. Filho and P. C. Treleaven, "Genetic-Algorithms Programming Environments," *Computer*, vol. 27, pp. 28-43, June 1994.
- [26] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
- [27] D. B. Fogel, "An Introduction to Simulated Evolutionary Optimization," *IEEE trans. on Neural Nets*, Jan. 1994.
- [28] R. Bianchini and C. Brown, "Parallel Genetic Algorithms on Distributed-Memory Architectures," Tech. Rep. TR 436, Computer Science Department, Univ. of Rochester, Rochester, 1992.
- [29] S. Baluja, "Structure and Performance of Fine-Grain Parallelism in Genetic Search," in *Proceeding of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, pp. 155-162, Univ. of Illinois Urbana-Champaign, July 1993.
- [30] B. P. Zeigler and J. Kim, "Asynchronous Genetic Algorithm on Parallel Computer," in *Proceeding of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, pp. 660-660, Univ. of Illinois Urbana-Champaign, July 1993.
- [31] K. DeJong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. dissertation, Univ. of Michigan, Ann Arbor, Michigan, 1975.

- [32] K. Deb and D. E. Goldberg, "Analyzing Deception in Trap Functions," *Foundations of Genetic Algorithms*, pp. 93-108, 1993.
- [33] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms," in *Proceeding of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, pp. 56-64, Univ. of Illinois Urbana-Champaign, July 1993.
- [34] M. Mitchell, S. Forrest, and J. H. Holland, "The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance," in *Proceeding of the First European Conference on Artificial Life*, Cambridge, MA, MIT Press/Bradford Books, 1992.
- [35] J. H. Holland, "Royal Road Functions," *Internet Genetic Algorithms Digest*, vol. 7, August 1993.
- [36] D. E. Goldberg, "First Flights at Genetic-Algorithm Kitty Hawk," Tech. Rep. IlliGal Report No. 94008, University of Illinois, General Engineering Department, Urbana-Champaign, IL, 1994.
- [37] J. S. Jang, "Self-learning fuzzy controller based on temporal back-propagation," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, 1992.
- [38] J. S. Jang, "ANFIS: Adaptive-network-based Fuzzy Inference Systems," *IEEE Transactions on Systems, Man Cybernetics*, vol. 23, no. 3, pp. 665-685, 1993.
- [39] M. J. Vasconcelos and B. P. Zeigler, "Simulation of forest landscape response to fire disturbances," *Ecological Modeling*, vol. 65, pp. 177-198, 1993.
- [40] C. J. Luh and B. P. Zeigler, "Abstracting Event-based Control Models for High Autonomy Systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 1, pp. 42-54, 1993.
- [41] W. H. Green and G. A. Ampt, "Studies on Soil Physics, I. The Flow of Air and Water through Soils," *Journal Agr. Sci.*, vol. 4, no. 1, pp. 1-24, 1911.
- [42] W. J. Rawls, J. J. Stone, and D. L. Brakensiek, "USDA-Water Erosion Prediction Project: Hillslope Profile Model Documentation," in *NSERL Rep. No. 2* (L. J. Lane and M. A. Nearing, eds.), ch. 4.1-4.12, West Lafayette, Ind.: National Soil Erosion Research Laboratory, USDA-ARS, 1989.

- [43] V. L. Lopes, "A Distributed Model of Stormflow and Sediment Yield for Small Watersheds," *Journal of Soil and Water Conservation*. (in press).
- [44] W. J. Rawls, D. L. Brakensiek, and K. E. Saxton, "Estimation of soil water properties," *Trans. ASAE Spec. Ed.; Soil and Water*, vol. 25, pp. 1316-1320.
- [45] S. T. Chu, "Infiltration During an Unsteady Rain," *Water Resources Research*, vol. 14, no. 3, pp. 461-466, 1978.
- [46] J. J. Stone, R. H. Hawkins, and E. D. Shirley, "Approximation Form of Green-Ampt Infiltration Equation," *ASCE Journal of Irrigation and Drainage Engineering*, vol. 120, no. 1, pp. 128-137, 1994.
- [47] B. P. Zeigler, "DEVS Representation of Dynamical Systems: Event-based Intelligent Control," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 72-80, 1989.
- [48] D. A. Woolhiser, R. E. Smith, and D. C. Goodrich, "KINEROS, A Kinematic Runoff and Erosion Model: Documentation and User Manual." U.S. Department of Agriculture, Agricultural Research Service, ARS-77, 1990.
- [49] P. K. Davis, "An Introduction to Variable-Resolution Modeling," *Naval Research Logistics*, vol. 42, pp. 151-181, 1995.
- [50] H. Praehofer, F. Auernig, and G. Reisinger, "An Environment for DEVS-Based Multiformalism Simulation in Common Lisp/CLOS," *Discrete Event Dynamic Systems: Theory and Applications*, 1993.
- [51] B. P. Zeigler, Y. Moon, D. Kim, and J. G. Kim, "DEVS-C++: An Environment for High Performance Simulation," in *Proceedings of the Hawaii International Conf. on Systems Science*, Hawaii, Jan. 1996.
- [52] M. A. Lee and H. Takagi, "Embedding Apriori Knowledge into an Integrated Fuzzy System Design Method Based on Genetic Algorithms," in *Proceeding of the 5th IFSA World Congress*, pp. 1293-1296, 1993.
- [53] B. P. Zeigler and J. W. Kim, "Event-based Fuzzy Logic Control System," in *Proceedings of 8th IEEE International Symposium on Intelligent Control*, Columbus, OH, pp. 611-616, 1993.

-
- [54] L. C. Schooley, F. E. Cellier, B. P. Zeigler, M. M. Marefat, and F. Wang, "High Autonomy Control of Space Resource Processing Plants," *Control Systems Magazine*, June 1993.

MISSION OF ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

- a. Conducts vigorous research, development and test programs in all applicable technologies;
- b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;
- c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;
- d. Promotes transfer of technology to the private sector;
- e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.